uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática
Curso 2017-2018

Trabajo Fin de Grado

"Diseño de un Procesador de Aplicación Específica (ASIP) para la Generación de Patrones"

Sergio Moreno García

Tutor:

Luis Alfonso Entrena Arrontes

Tabla de Contenidos:

Listado de Figuras:

Listado de Tablas:

1. Introducción

El objetivo de este trabajo es el diseño e implementación de un Procesador de Aplicación Específica (ASIP) para la generación de patrones.

Un ASIP es un procesador con un conjunto de instrucciones de uso específico, es decir, diseñado a medida para cumplir la función requerida de la manera más eficiente posible, siendo órdenes de magnitud más rápido que un procesador de uso general.

En este caso el ASIP cumple la función de generación de patrones de ondas digitales, para su uso en una variedad de aplicaciones, entre las que se incluyen sistemas de control de electrónica de potencia, control de sensores y actuadores matriciales, etc.

Los patrones se generan mediante la salida de datos durante un tiempo determinado. Se dispone de una única instrucción de movimiento de datos, que escribe un dato durante un tiempo determinado en la salida, siendo el dato y el tiempo indicados por la instrucción, y contenidos en la memoria interna del ASIP. Además de esta función, hay numerosas instrucciones de control que permiten crear bucles y llamadas a subrutinas para la programación de patrones complejos.

El ASIP debe ser altamente configurable, por lo que se ha diseñado de manera genérica, es decir, su estructura y comportamiento depende de una serie de parámetros configurables por el usuario.

1.1 Motivación del Proyecto

La historia de la computación está inherentemente ligada al desarrollo del diseño electrónico digital. Sistemas computacionales más complejos han incrementado la investigación de nuevas tecnologías electrónicas.

La potencia de computación ha aumentado exponencialmente, como es descrito por la Ley de Moore: Cada dos años se duplica el número de transistores en circuitos integrados típicos. Actualmente, el tamaño mínimo de canal semiconductor que podemos producir es de 10nm, con tecnología de 7nm en desarrollo. Sin embargo, según nos acercamos al límite físico del tamaño mínimo de un canal semiconductor este crecimiento deja de ser exponencial y se acerca asintóticamente a este límite. Debido a esta limitación, se ha de buscar medidas alternativas para incrementar la eficiencia de un diseño electrónico.

El modelo más común de computación en las últimas décadas ha sido el diseño del microprocesador de carácter general. Este diseño consistía en un microprocesador con un conjunto de instrucciones predeterminado que fuese capaz de realizar una amplia gama de operaciones mediante combinaciones de estas instrucciones, siendo programado mediante software.

Debido a su carácter general, el modelo de procesador no está optimizado para operaciones complejas, por lo que es mejor usar una plataforma de hardware específica optimizada para la aplicación específica que se desea, incrementando en una gran medida la eficiencia del sistema.

Mediante el uso de ASICs (Application-Specific Integrated Circuits) o Circuitos Integrados de Aplicación Específica se aumenta la eficiencia, pero la especificidad de los circuitos aumenta mucho su coste, siendo una desventaja para su viabilidad económica. Sin embargo, los avances de las últimas décadas (lenguajes HDL, herramientas de síntesis, dispositivos lógicos programables, etc.) han permitido reducir el coste de los ASICs drásticamente, volviéndose estos la elección clave en sistemas que requieren alta eficiencia de computación.

El modelo más común es un modelo híbrido, en el que se dispone de un microprocesador que realiza las operaciones de carácter general del sistema, y un ASIC que se encarga de procesar las operaciones más computacionalmente pesadas. Debido al desarrollo tecnológico y la disminución de costes de los ASICs, esta opción se está volviendo más común, de tal manera que muchos dispositivos anteriormente usados como periféricos ahora cuentan con su propio circuito de pre-procesado de su información, reduciendo la carga sobre el sistema central. Un ejemplo es el de satélites (de comunicación, meteorológicos, de GPS, etc.) que antes funcionaban meramente como sensores y tan solo retransmitían su información a un sistema de procesado en la superficie del planeta, mientras que ahora cuentan con su propio Hardware que les permite procesar su información antes de transmitirla.

Esta tendencia a la especificidad del hardware ha sido promovida también por la tecnología de dispositivos lógicos reprogramables, como las FPGAs, que permiten una rápida creación de

prototipos y facilitan el diseño del hardware, siendo a veces más económicamente viables que un ASIC (debido a su fabricación en masa).

Una mejora reciente sobre el modelo procesador-ASIC es el uso de ASIPs. Un ASIP (Application Specific Instruction-Set Processor) es un Procesador cuyo Conjunto de Instrucciones está hecho a medida para una aplicación específica. El ASIP puede ser programado desde el procesador para cada ejecución, por lo que tiene una funcionalidad muy flexible manteniendo su rendimiento, si bien es menos eficiente que un ASIC específico para cada uso. Es el punto medio entre la flexibilidad de un Microprocesador de Uso General y un ASIC.

Un ejemplo típico de este sistema es un ordenador personal, que dispone, entre otros dispositivos, de un microprocesador central y una tarjeta gráfica. De esta manera el microprocesador central realiza todas las operaciones menos el procesado gráfico, que queda delegado a la tarjeta gráfica, un ASIC optimizado para esas operaciones concretas.

Debido a estos avances, es necesario buscar soluciones de hardware específico para problemas de computación.

1.2 Objetivos Detallados del Proyecto

El objetivo principal del proyecto es el diseño del ASIP y su implementación en un medio físico, así como la validación de su comportamiento mediante una serie de pruebas. Las etapas del desarrollo para alcanzar este objetivo final son las siguientes:

1. **Diseño del sistema e implementación en HDL:**
   Se deberá desarrollar un diseño general del ASIP, su diagrama de bloques, sus entradas y salidas, su comportamiento, etc. Una vez se haya diseñado el ASIP, se deberá implementar mediante un HDL.

   Los HDLs o Lenguajes de Descripción de Hardware son lenguajes de modelado que describen la estructura y el comportamiento de un circuito electrónico, normalmente de carácter digital. Se discuten en más detalle en el apartado 2. El lenguaje usado será VHDL, el HDL estándar europeo.

2. **Desarrollar una aplicación software para facilitar la programación del ASIP.**
   Se deberá desarrollar un entorno de desarrollo para el ASIP que consista en un programa ensamblador que traduzca los programas del ASIP, escritos en el lenguaje de máquina propuesto por el tutor, a código binario que pueda ser directamente procesado por el ASIP. De esta manera se facilitara el diseño de programas para el ASIP.

3. **Validación del diseño mediante simulación:**
   Para validar el correcto funcionamiento del diseño HDL se realizarán simulaciones del comportamiento del mismo en un Entorno de Desarrollo de Hardware.
   Se considerarán distintos casos de prueba para validar todas las funciones y posibles errores del diseño.

4. **Síntesis e Implementación. Pruebas Físicas.**
   Se realizará la síntesis e implementación del diseño en un medio físico mediante un Entorno de Desarrollo de Hardware. Como medio físico se ha escogido el dispositivo SoC ZYNQ-7000 de Xilinx. Este dispositivo se compone principalmente de un microprocesador de arquitectura ARM y una FPGA conectadas mediante un bus AMBA. Se describe en más detalle en el apartado 7.

   El diseño del ASIP se implementará en la FPGA con valores por defecto de los parámetros genéricos, y el programa se cargará en el microprocesador. Se deberá desarrollar la integración correspondiente de los dos componentes, es decir, la interfaz de configuración del ASIP con el bus AMBA.

   De nuevo, se deberán considerar programas de prueba que verifiquen las distintas funcionalidades del diseño.

2. Contexto del Trabajo y Tecnologías Utilizadas

2.1 Contexto del Trabajo.

2.1.1 Procesadores de Conjunto de Instrucciones de Aplicación Específica. ASIP.

Definición.

Un ASIP o Procesador de Conjunto de Instrucciones de Aplicación Específica (por sus siglas en inglés, Application Specific Instruction-Set Processor), es un componente electrónico digital basado en una arquitectura ISA (Instruction Set Architecture), es decir, su comportamiento depende de un conjunto de instrucciones, de la misma manera que un Microprocesador de Uso General (GPP, General Purpose Processor), es decir, el ASIP es una plataforma de hardware programable.

Sin embargo, a diferencia de un GPP, su lógica sólo soporta un conjunto reducido de aplicaciones, es decir, un ASIP es un diseño a medida, específico para una aplicación o conjunto de aplicaciones. Esta especificidad confiere al ASIP una gran ventaja en rendimiento al coste de flexibilidad [1]. Por otro lado, los ASICs o Circuitos Integrados de Aplicación Específica (Application-Specific Integrated Circuit) representan el otro extremo del espectro, teniendo la máxima especificidad (y por tanto rendimiento) al coste de ser completamente inflexible. De esta manera el ASIP cubre el punto intermedio del espectro, teniendo un buen equilibrio entre flexibilidad y rendimiento.

Debido a la posición central del ASIP en el espectro de especificidad de un diseño electrónico, podemos encontrar muchos tipos de arquitecturas, dependiendo del énfasis del diseño en rendimiento o flexibilidad. Al nivel de arquitectura general, podemos distinguir dos categorías de ASIP:

- **Basados en ISA (Instruction Set Architecture):** Estos ASIPs siguen el diseño tradicional ISA o Arquitectura de Conjunto de Instrucciones de la misma manera que un GPP. Se diseña una plataforma hardware fija, basada en un Conjunto de Instrucciones y se comercializa. La aplicación del ASIP a distintas aplicaciones consistirá en programarlo mediante software.

- **Basados es Hardware Programable:** Estos ASIPs siguen el diseño de lógica programable similar a dispositivos como las FPGAs. Cuentan con una pequeña arquitectura ISA fija, a la que se añade arquitectura para soportar un conjunto de instrucciones adicionales diseñadas por el usuario. La aplicación del ASIP a distintas aplicaciones consiste principalmente en diseño hardware.

Esta clasificación no es muy estricta debido a la tendencia actual de incluir procesadores y hardware programable en un solo diseño.

El ASIP diseñado en este proyecto es del tipo ISA, y por tanto sólo consideraremos ésta arquitectura a la hora de tomar decisiones de diseño e implementación.

En el diseño de un ASIP, hay que tener en cuenta ciertos factores. La interfaz del ASIP con los sistemas externos define la comunicación entre el ASIP y el resto de la arquitectura. Esto también puede afectar el rendimiento del sistema, por ejemplo, si la entrada de datos no puede procesar suficientemente rápido ocurrirá un cuello de botella debido a la interfaz del sistema.

De manera similar, las etapas de ejecución del ASIP deben estar diseñadas para optimizar el rendimiento de la arquitectura completa y no solo del ASIP. Las etapas tradicionales de una maquina RISC (Lectura, Decodificación, Ejecución y Escritura) pueden no ser las óptimas para la aplicación [2].

También se deberá tener en cuenta otros aspectos como el conjunto de instrucciones del ASIP, el número de elementos procesadores, las operaciones que realizará, los registros y memorias internas, etc.

Toda decisión tomada respecto al diseño del ASIP tendrá un balance de rendimiento y flexibilidad que el diseñador deberá elegir en base a las necesidades de la arquitectura completa.

Ventajas del ASIP frente a otras arquitecturas

Al diseñar un sistema de aplicación específica o un sistema embebido, se suele usar una o más de las siguientes arquitecturas:

- **GPP:**
  La funcionalidad del sistema se implementa exclusivamente en software. La flexibilidad es máxima, pero el GPP no está optimizado para rendimiento, consumo, coste, espacio físico o disipación de calor.

- **ASIC:**
  Los ASICs consisten del extremo opuesto del espectro al GPP. Cuentan con rendimiento y consumo óptimos, pero al coste de flexibilidad. Debido a la dificultad del ASIC de realizar tareas diferentes a su aplicación específica, suelen encontrarse en un sistema complejo junto a un GPP que realiza las tareas más generales y menos demandantes.

- **ASIP:**
  El ASIP consiste del punto intermedio entre un GPP y un ASIC. Los ASIP cuentan con flexibilidad y rendimientos medios. Los ASIPs abarcan la zona intermedia del espectro, por lo que es posible encontrar ASIPs más específicos, cercanos a un ASIC, y ASIPs más generales, cercanos a un GPP, en función del factor determinante de la aplicación requerida: rendimiento o flexibilidad.
  Debido a esta variedad de opciones y bajo precio los ASIPs son perfectos para uso en sistemas embebidos y soluciones SoC (System-On-Chip).

En la Figura 1 podemos observar una rápida comparación de la flexibilidad frente al rendimiento para los tres tipos de arquitecturas.

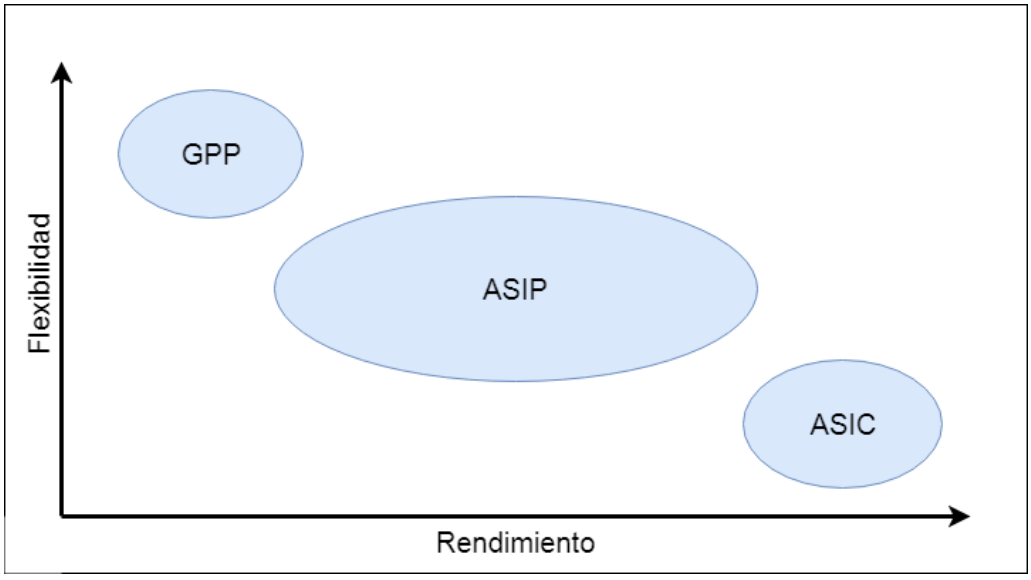

Figura 1: Comparación del Rendimiento y Flexibilidad de las distintas Arquitecturas.

En la Tabal 1 podemos observar una comparación en detalle de las características de las tres arquitecturas, incluyendo su rendimiento, flexibilidad, diseño hardware y software requerido para desarrollar la aplicación, consumo, reusabilidad y demanda comercial y coste.

	GPP	ASIP	ASIC
Rendimiento	Bajo	Alto	Muy Alto
Flexibilidad	Excelente	Buena	Pobre
Diseño Hardware	Ninguno	Alto	Muy Alto
Diseño Software	Bajo	Alto	Ninguno
Consumo	Alto	Medio	Bajo
Reusabilidad	Excelente	Buena	Pobre
Demanda	Muy Grande	Grande (dependiendo en la aplicación)	Pequeño
Coste	Hardware barato, el coste radica en el desarrollo Software	Depende del sistema donde se implemente.	Depende del volumen de producción.

Tabla 1: Comparación de las Características de las distintas Arquitecturas.

Aplicaciones

Los ASIP suelen ser utilizados para procesado de señales digitales o codificación de video. Los diseños tradicionales para estas aplicaciones son inflexibles, y es difícil reutilizar el hardware para aplicaciones similares. La flexibilidad de los ASIPs permite al diseñador reutilizar el mismo componente para una amplia selección de aplicaciones similares, manteniendo un rendimiento elevado.

Los ASIP suelen ser exportados como un IP (Intelectual Property), una unidad de diseño que puede ser incluida como un componente dentro de un sistema más complejo, usualmente un SoC (System on a Chip), para acelerar ciertas funciones del sistema que son poco eficientes en el procesador central. Se hablará más a fondo de los IPs en el apartado 2.1.2

La aplicación más común de los ASIP como sistemas embebidos es la de D**SP (Digital Signal Processing)**, usado para acelerar ciertas operaciones de una aplicación mediante el uso de hardware a medida (MACs, FFTs, etc.). A continuación se listan algunas de las aplicaciones de los ASIPs de tipo DSP:

- **Imagen y Video:**
  - Control de Sensores (cámaras).
  - Control de Actuadores (pantallas).
  - Transmisión de Datos (conexiones HDMI, VGA, etc.).
  - Procesado de Imagen y Video (Codificación de video e imagen).
- **Procesado Gráfico:**
  - Gráficos 2D.
  - Gráficos 3D.
  - Radar, ultrasonido, etc.
- **Aprendizaje (Machine Learning):**
  - Redes Neuronales.
  - Procesado de Lenguaje Natural.
  - “Deep Learning”.

2.1.2 Bloques de Propiedad Intelectual. IPs.

Un Bloque de Propiedad Intelectual o "IP Core" es una unidad de lógica reutilizable o un diseño esquemático de un circuito integrado que es propiedad intelectual de una sola persona o empresa. Los IPs pueden describirse como el equivalente a una librería de software para el diseño hardware. Los Bloques IP suelen usarse como componentes en el diseño de ASICs y en diseños electrónicos basados en FPGAs.

Podemos diferenciar dos tipos claramente diferenciables de IPs en función de su nivel de abstracción:

- **"Soft Cores":** Estos IPs son diseños en código RTL sintetizable, normalmente presentados como código HDL (definido más adelante en el apartado 2.2.1). Esto permite al usuario un alto nivel de modificación y adaptabilidad del IP a su aplicación específica, aunque los proveedores de IP no suelen ofrecer soporte para diseños modificados.
  Algunos IPs de este tipo son diseños previamente sintetizados en formato de "netlist", es decir, una lista de sus componentes individuales (puertas lógicas genéricas) y las conexiones entre ellos. Estos IPs también proporcionan una mejor protección contra ingeniería inversa, protegiendo la inversión del diseñador.

- **"Hard Cores":** Estos IPs son diseños previamente sintetizados e implementados, es decir, se componen de un esquema y colocación física de los componentes. Este tipo de diseño impide la modificación del mismo, pero ofrece una mejor predictibilidad en términos de área, consumo y rendimiento finales del IP. Los "Hard IP" suelen ser usados más comúnmente en diseños analógicos y mixtos (como los PLLs, DACs, ADCs, etc.), aunque también se encuentran en diseños con funcionalidades fijas, como codificación MP3, codificación de video digital, o DSPs como los FFTs.

Comercialización de IPs:

Los desarrolladores de IPs varían en tamaño desde individuos hasta grandes empresas, localizadas por todo el mundo.

"Silicon Intellectual Property" o Silicon IP es un modelo de negocio en el que la compañía desarrolla diseños electrónicos que en vez de fabricarse son vendidos como IPs a otras compañías bajo licencias de uso. Este modelo incluye grandes compañías como ARC Internacional y ARM Holdings.

Ejemplos comunes de IPs de uso extendido incluyen microprocesadores de arquitecturas ARM o MIPS y controladores de periféricos comunes como USB o Ethernet.

2.2 Tecnologías Utilizadas

2.2.1 Lenguajes de Descripción de Hardware

Los Lenguajes de Descripción de Hardware o HDLs (Hardware Description Languages) son lenguajes de programación especializados para el desarrollo de circuitos electrónicos, principalmente digitales, que se utilizan para definir la estructura y operación de dichos circuitos. Los HDLs hacen posible la descripción formal de un circuito electrónico, lo que posibilita un análisis y simulación automáticos, así como facilita la integración automática de distintos circuitos.

Los HDLs son similares a otros lenguajes de programación como C++ o Java, teniendo similares estructuras de control y expresiones, sin embargo tienen una importante diferencia: Los HDLs son lenguajes de modelaje.

Mientras que un lenguaje de programación convencional consiste en una serie de operaciones a ejecutar por el procesador de un ordenador, los HDLs describen una estructura de componentes físicos y su comportamiento asociado. Por lo tanto los HDLs no tienen una orden de instrucciones, sino que sus expresiones son concurrentes, es decir, todas ocurren a la vez y continuamente, su valor dependiendo de otras expresiones y del tiempo. Si bien hay lenguajes de programación concurrentes que actúan también de esta manera, su principal diferencia con los HDLs es que éstos incluyen de manera explícita la noción del tiempo, pues es una característica real en los circuitos reales que representan.

Los HDLs describen los circuitos electrónicos al nivel de abstracción RTL (Register-Transfer Level), es decir, como un flujo de señales digitales entre una serie de registros, así como las operaciones lógicas aplicadas a dichas señales. Este nivel abstracción de añadido permite al ingeniero diseñar circuitos a más alto nivel que el diseño clásico por esquemas, habilitando el desarrollo de circuitos más extensos y complejos que no habían sido posibles previamente debido a su escala y complejidad, como los diseños VLSI (Very Large Scale Implementation) que integran millones de transistores en un único chip.

Los HDLs pueden ser compilados con Herramientas de Síntesis Lógica que generan archivos RTL que consisten de una serie de “netlists” (conjunto de componentes electrónicos y las conexiones entre ellos) que puede ser directamente manufacturada en un ASIC (Application Specific Integrated Circuit) o Circuito Integrado de Aplicación Especifica. De esta manera el HDL incrementa exponencialmente la productividad del diseño, automatizando completamente varios niveles de abstracción.

Los archivos RTL compilados también pueden usarse para configurar un PLD (Programmable Logic Device) o Dispositivo Lógico Programable, facilitando en gran medida la creación de prototipos de hardware específico.

2.2.2 Síntesis Lógica

La Síntesis Lógica es un proceso mediante el cual se procesa un diseño abstracto de un circuito, normalmente a nivel RTL (Register-Transfer Level) se traduce en su implementación física, es decir, un conjunto de puertas lógicas. Un ejemplo típico de uso de Síntesis Lógica es la síntesis de lenguajes de HDL como VHDL y Verilog. Algunas herramientas de Síntesis Lógica generan "bitstreams" (secuencias binarias) para implementar estos diseños en FPGAs, mientras que otras están orientadas a la fabricación de ASICs.

El primer paso de la síntesis es el diseño lógico, en el que se traduce el diseño funcional a una representación de las operaciones lógico-aritméticas y el flujo de datos, normalmente en formato RTL. Ejemplos de estas operaciones lógicas son las operaciones booleanas o binarias, que son las más básicas, siendo operaciones más complejas la adición o la multiplicación. Esta operación suele estar optimizada, las herramientas aplican algoritmos de simplificación para obtener la expresión lógica más reducida posible.

El siguiente paso es la síntesis de la estructura física de los componentes, es decir, su colocación espacial y el emplazamiento de sus conexiones, también conocido como "Place & Route". Este paso optimiza la colocación de tal manera que se reduzca el consumo y aumente la frecuencia de operación del circuito.

Varias herramientas de síntesis facilitan la implementación de estos diseños en una plataforma de desarrollo como una FPGA, permitiendo la asignación de señales a pines externos y la programación y depuración del hardware en la FPGA.

Finalmente, algunas herramientas de síntesis permiten la síntesis a alto nivel (HLS High Level Synthesis), que permiten sintetizar circuitos a partir de lenguajes de alto nivel como C o C++.

2.2.3 Dispositivos Lógicos Programables

Los Dispositivos Lógicos Programables (PLDs, Programmable Logic Devices) son componentes electrónicos usados para construir circuitos digitales reconfigurables. A diferencia de un circuito lógico tradicional, un PLD no tiene una función lógica definida tras su fabricación, sino que debe ser programado para reconfigurar su circuito en una función lógica determinada.

La principal ventaja de estos dispositivos es un prototipado rápido y barato de diseños electrónicos digitales, permitiendo generar el nuevo circuito a partir de los últimos cambios en cuestión de minutos. Debido a su popularidad, en algunos casos (principalmente debido a un bajo volumen de producción) es incluso más económico implementar el diseño en un PLD en vez de en un ASIC.

Existe una variedad de tecnologías PLD, entre las que se encuentran PLAs, PALs, GALs, y CPLDs entre otras. Su principio de funcionamiento es principalmente una matriz de puertas lógicas AND y OR, resultando en una suma de productos o un producto de sumas. También cuentan con una serie de biestables para almacenar información. La programación del dispositivo determina cual se escoge entre ellas, determinando la función lógica del PLD.

Los PLDs modernos están basados en tecnología FPGA (Field-Programmable Gate Array) o Matriz de Puertas Programables. Las FPGAs también cuentan con una serie de registros interconectados con bloques lógicos, pero la diferencia principal en arquitectura es la implementación de estos bloques lógicos. Mientras que en los PLDs tradicionales los bloques lógicos consisten en un conjunto de puertas lógicas, siendo la función resultante una suma de productos, las FPGAs tienen en su lugar LUTs (Look-up Tables) o tablas de consulta. Las LUTs son pequeñas memorias conectadas de tal manera que las señales de entrada determinan la dirección de lectura de la LUT, que será la salida del bloque lógico. Con esto se implementa la función lógica con una tabla de verdad física. Los datos de esta tabla de verdad son los datos que se programan externamente, determinando el circuito de la FPGA. Las FPGAs también cuentan con una serie de circuitos predeterminados no-reconfigurables o “hardwired” de uso extendido como sumadores, multiplicadores, PLLs, y memorias internas de varios tipos.

Actualmente las FPGAs más avanzadas son parte de un dispositivo más complejo llamado SoC (System on a Chip). Estos circuitos integrados suelen contener una FPGA, un microprocesador y ciertos otros componentes como bloques de memoria FLASH, ROM, etc. De esta manera se puede realizar cualquier diseño de electrónica digital como una aplicación empotrada en un microprocesador con rendimiento acelerado mediante hardware a medida. Un ejemplo de SoC es la placa Xilinx ZYNQ-7000 All-Programmable SoC que será usada en este proyecto, la cual se compone de un microprocesador ARM y una FPGA, así como numerosos periféricos.

3. Requerimientos y Especificaciones del Diseño

3.1 Descripción Funcional del Diseño

La función principal del procesador es la de emitir datos, que llamaremos patrones de aquí en adelante, durante un tiempo específico, que llamaremos retardo. Para esto el procesador tiene dos bloques funcionales básicos:

- **Generador de Eventos:**
  Este bloque recibe instrucciones para escribir un patrón en la salida durante un retardo específico.

  Los patrones y retardos a usar en el programa deben ser previamente almacenados en memoria interna. Así pues, las instrucciones meramente señalan a una posición de cada una de estas dos memorias.

  Por último, se considera un sistema de prescalado. Durante la ejecución, todos los retardos se multiplican por este valor de prescalado, pudiendo así efectivamente alterar la base de tiempos de la ejecución de los eventos. El valor del prescalado también debe ser guardado en memoria interna antes de la ejecución del programa.

- **Generador de Secuencias:**
  Este bloque se encarga de controlar todos los saltos, o "branching" del programa, es decir: bucles, saltos incondicionales y llamadas a subrutinas. Su funcionamiento consiste en leer la instrucción actual de la memoria interna de instrucciones, identificar su tipo y sus parámetros y actuar de acuerdo a ellos.

  En el caso de instrucciones para escribir patrones a la salida, estas se envían al Generador de Eventos. Las instrucciones de control no se mandan al Generador de Eventos, sino que simplemente ejecutan su función dentro del Generador de Secuencias, es decir, se salta a una nueva dirección de memoria cuando se corresponda, y se administran los "niveles de llamada" (un indicador de instrucciones anidadas en bucles o subrutinas) cuando la instrucción es de bucle o llamada a subrutina.

  Para administrar los niveles de llamada se cuenta con una Pila o memoria LIFO. Cuando el programa entra en un bucle o llama a una subrutina, se introduce en la Pila un nuevo registro, al que llamaremos Nivel de Llamada de aquí en adelante, que contiene la dirección siguiente a la instrucción actual y el número de iteraciones restantes. Se distinguen dos funcionamientos diferentes entre bucles y subrutinas:

    - **Bucles:** Se ejecutan las instrucciones siguientes al bucle hasta llegar a una instrucción de retorno. Ésta disminuye en uno el número de iteraciones restantes del nivel de llamada actual y el programa salta a la dirección de retorno del nivel de llamada. En el caso de que no queden iteraciones restantes, se elimina ese nivel de llamada y se continúa con la siguiente instrucción a la de retorno

- **Llamadas a Subrutinas:** El programa salta a la primera instrucción de la subrutina y continúa hasta llegar a una instrucción de retorno. Ésta disminuye en uno el número de iteraciones restantes del nivel de llamada actual y el programa salta al comienzo de la subrutina. En caso de que no queden iteraciones restantes, se elimina ese nivel de llamada y se continúa con la dirección de retorno del nivel de llamada, es decir, la instrucción siguiente a la llamada a subrutina.

El caso de bucles o llamadas a subrutinas se soluciona si la memoria de niveles de llamada es de tipo LIFO o pila. De esta manera el nivel de llamada actual es siempre el más anidado, y al resolverlo se vuelve al estrictamente superior.

Además de esta función básica, el ASIP debe implementar las siguientes funcionalidades añadidas:

- **Control Externo:**
  El ASIP debe implementar un sistema de control mediante entradas externas. Se realiza el control mediante la instrucción SYNC, descrita más adelante, que para la ejecución del ASIP hasta que una de las entradas sea igual a un valor concreto, siendo la entrada y el valor especificados en la instrucción SYNC.

- **Interfaz de Entrada y Salida:**
  El ASIP debe contar con los siguientes puertos:
  - **Salida de Datos (Salida):** Puerto por el que el ASIP emite los patrones. Su ancho es igual al de los patrones.
  - **Puerto de Control Externo (Entrada):** Conjunto de entradas externas de un bit que se usan en el Control Externo.
  - **Puerto de Escritura (Entrada)**: Interfaz de escritura usada para cargar los datos de programa (instrucciones, patrones, retardos y prescalado) en el ASIP antes de la ejecución.

3.2 Descripción Técnica del Diseño

3.2.1 Arquitectura

El diagrama de bloques general (simplificado) del diseño se puede observar en la Figura 2.

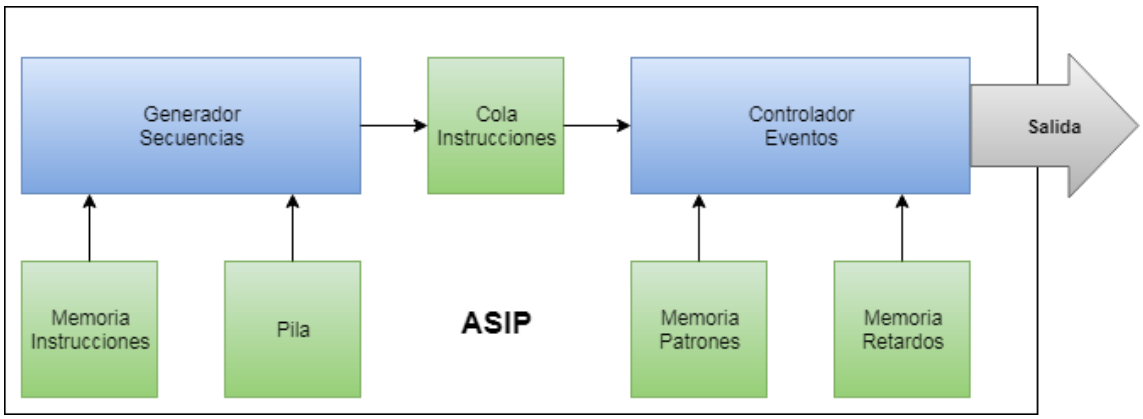

Figura 2: Diagrama de Bloques general del Diseño Original.

Se pueden distinguir los siguientes bloques:

En color verde observamos los Bloques de Memoria, de los que hay cinco:

- **Memoria de Instrucciones**: Este bloque almacena el programa a ejecutar por el ASIP. Cuenta con los siguientes elementos:
  - **Memoria de Programa (PM):** Memoria RAM que almacena las instrucciones del de programa.
  - **Contador de Programa (PC):** Contador que indica la dirección de la PM (instrucción) a ejecutar.
  - **Registro de Instrucción (IR):** Registro que almacena la instrucción actual a ejecutar.

- **Pila:** Este bloque es una memoria de tipo LIFO (Last-In First-Out) o Pila, donde se almacenará la información correspondiente a los niveles de llamada. Cuenta con los siguientes elementos:
  - **Pila (Stack):** Memoria LIFO que almacena los datos de los distintos niveles de llamada.
  - **Registro Top-Of-Stack (TOS):** Registro que almacena el nivel de llamada actual, y que funciona como un registro más colocado en el tope de la Pila, cuyo valor se puede modificar externamente.

- **Memoria de Patrones:** Este bloque consiste en una memoria RAM que almacena los patrones a usar en el programa que más tarde se observan en la salida.

- **Memoria de Retardos:** Este bloque consiste en una memoria RAM que almacena los retardos a usar en el programa que más tarde se observan en la salida. El bloque incluye un Registro de Prescalado, donde se guarda el valor de Prescalado del Reloj.

- **Cola de Eventos:** Este bloque es una memoria de tipo FIFO (First-In First-Out) o cola, donde se almacenan las instrucciones a ejecutar por el bloque Generador de Eventos.

En color azul observamos los Bloques Lógicos, de los que hay 2:

- **Generador de Secuencias:** Este bloque administra los niveles de llamada. El bloque lee las instrucciones de la Memoria de Instrucciones y las procesa, distinguiendo dos tipos:
    - **Instrucciones de Generación de Eventos**: Se escriben en la Cola de Eventos para su futura ejecución.
    - **Instrucciones de Control**: Las instrucciones de control o "branching", como los bucles o las llamadas a subrutinas, crean o eliminan niveles de llamada, que se almacenan en el bloque Pila, y mueven el Contador de Programa a la nueva línea correspondiente. Estas instrucciones son ejecutadas directamente por el Generador de Secuencias y no se escriben en la Cola de Eventos.

  Se considera un "pipelining" o procesado por etapas (en cadena) en las instrucciones: Las instrucciones son procesadas en las siguientes dos etapas:
    - **Etapa de Búsqueda:** Se lee la instrucción actual de la memoria y se guarda en el Registro de Instrucción (IR).
    - **Etapa de Procesado:** El Generador de Secuencias procesa la instrucción del IR de la manera anteriormente descrita.

  La Etapa de Búsqueda se aprovecha para aumentar la eficiencia debido a que se busca la instrucción siguiente a la actual en el mismo ciclo en el que se ejecuta ésta. Se debe tener en cuenta el retraso de un ciclo de reloj entre etapas a la hora de procesar las instrucciones, dándose el caso de que en algunas Instrucciones de Control se debe descartar la instrucción en la Etapa de Búsqueda, pues debido al salto de línea ésta no será la siguiente instrucción a procesar.

- **Generador de Eventos:** Este bloque se encarga de producir la salida de datos. Lee las Instrucciones de Generación de Eventos de la Cola de Eventos, cada una de las cuales indica un par de direcciones de memoria, una apuntando a la memoria de patrones y otra a la memoria de retardos. El Controlador lee el patrón y el retardo correspondientes de la memoria, así como el valor del prescalado, y escribe el patrón en la salida durante un tiempo igual al retardo multiplicado por el valor del prescalado, tras el cual lee una nueva instrucción de la cola y repite el proceso.

3.2.2 Parámetros Genéricos

El diseño debe ser genérico para poder adaptarse a distintas aplicaciones, es decir, deben poderse configurar ciertos parámetros como el tamaño de las memorias, el ancho de los datos de salida, etc. Los parámetros del diseño que deben ser configurables se describen en la Tabla 2, junto con su valor por defecto.

Parámetro	Valor por Defecto	Descripción
NPatrones	256	Número de direcciones de memoria en la Memoria de Patrones
NSalidas	32	Ancho de bits de los patrones y la salida de datos
NRetardos	64	Número de direcciones de memoria en la Memoria de Retardos
AnchoRetardo	20	Ancho de bits de los retardos
NInstrucciones	8192	Número de direcciones de memoria en la Memoria de Instrucciones
AnchoInstrucción	16	Ancho de bits de las instrucciones
TamañoPila	8	Número de direcciones de memoria de la Pila (sin contar el Registro TOS)
MaxIteraciones	4096	Valor máximo que puede tomar el número de iteraciones de un nivel de llamada.
NEntradas	16	Ancho de bits de la entrada externa.
MaxPrescalado	10	Valor máximo del prescalado
TamañoCola	16	Número de direcciones de la Cola de Eventos

Tabla 2: Parámetros Genéricos configurables del Diseño Original.

Por tanto, el tamaño de las memorias y registros será el especificado en la Tabla 3.

Elemento	Tipo	Ancho de Bits		Número de Direcciones	
		Genérico	Por Defecto	Genérico	Por Defecto
Memoria de Instrucciones	RAM	AnchoInstrucción	16	NInstrucciones	8192
Memoria de Patrones	RAM	NSalidas	32	NPatrones	256
Memoria de Retardos	RAM	AnchoRetardo	20	NRetardos	64

Elemento		Tipo	Ancho de Bits		Número de Direcciones	
			Genérico	Por Defecto	Genérico	Por Defecto
Cola de Eventos		FIFO	AnchoInstrucción	16	TamañoCola	16
Pila	Dirección	LIFO	$\log_2 NInstrucciones$	13	TamañoPila	8
	Iteraciones		$\log_2 MaxIteraciones$	12		
Registro de Instrucción		Registro	AnchoInstrucción	16	1	
Registro de Patrón		Registro	NSalidas	32		
Registro de Retardo		Registro	AnchoRetardo	20		
Registro de Prescalado		Registro	$\log_2 MaxPrescalado$	4		
Contador de Retardo		Contador	AnchoRetardo	20		
Contador de Prescalado		Contador	$\log_2 MaxPrescalado$	4		
TOS	Dirección	Registro	$\log_2 NInstrucciones$	13		
	Iteraciones		$\log_2 MaxIteraciones$	12		

Tabla 3: Tamaños de las Memorias Internas.

3.2.3 Conjunto de Instrucciones

Se distinguen dos tipos de instrucciones:

- **Instrucciones de Generación de Eventos:** Estas instrucciones no se ejecutan directamente por el Generador de Secuencias, se escriben en la Cola de Eventos y son ejecutadas más adelante por el Generador de Eventos.
- **Instrucciones de Control:** Estas instrucciones lidian con la administración de los niveles de llamada, es decir, creación y resolución de bucles y llamada a subrutinas. Estas instrucciones son ejecutadas directamente por el Generador de Secuencias.

Todas las instrucciones tienen un ancho de 16 bits. Los primeros 2, 3 o 4 bits (dependiendo de la instrucción) serán un Código de Operación (OpCode) que determina la instrucción, mientras que el resto de bits componen los argumentos de la instrucción.

Las instrucciones provistas en el diseño inicial junto con su OpCode identificador y sus argumentos se pueden observar en la Tabla 4 (Instrucciones de Generación de Eventos) y en la Tabla 5 (Instrucciones de Control).

Instrucciones de Generación de Eventos:

Instrucción	OpCode	Argumentos		Funcionamiento
		Nombre	Tamaño	
MOV	00	IDPatrón	8	Es la instrucción principal del ASIP. Indica un par Patrón-Retardo a ejecutar por el Generador de Eventos
		IDRetardo	6	
SYNC	0110	Entrada	4	Es una instrucción de pausado que para la ejecución hasta que la entrada externa especificada concuerde con el valor indicado.
		Valor	1	

Tabla 4: Instrucciones de Generación de Eventos

Instrucciones de Control:

Instrucción	OpCode	Argumentos		Funcionamiento
		Nombre	Tamaño	
LOOP	0101	Count	12	Abre un nuevo nivel de llamada en la pila, cuya dirección de retorno es la siguiente a la actual, y su número de iteraciones es igual a Count.
LDC	0100	Count	12	Carga un número de iteraciones en la pila sin crear un nuevo nivel de llamada.

Instrucción	OpCode	Argumentos		Funcionamiento
		Nombre	**Tamaño**	
CALL	110	Addr	13	Abre un nuevo nivel de llamada en la pila, cuya dirección de retorno es la siguiente a la instrucción actual, y su número de iteraciones 1. El Contador de Programa salta a la dirección determinada por Addr.
JUMP	111	Addr	13	El Contador de Programa salta a la dirección determinada por Addr. No se crea un nuevo nivel de llamada.
DEC	100	N/A	N/A	Reduce el número de iteraciones del nivel de llamada actual en uno. Si quedan iteraciones pendientes, el Contador de Programa salta a la dirección de retorno del nivel de llamada actual, si no, se elimina el nivel de llamada actual y se prosigue con la siguiente instrucción.
DECRET	101	Addr	13	Reduce el número de iteraciones del nivel de llamada actual en uno. Si quedan iteraciones pendientes, el Contador de Programa salta a la dirección de retorno determinada por Addr, si no, se elimina el nivel de llamada actual y se salta a la dirección de retorno de éste.

Tabla 5: Instrucciones de Control

4. Diseño Final Implementado

4.1 Descripción Funcional del Diseño

Se han mantenido las mismas funciones básicas que el diseño original, así como ciertas mejoras al diseño y funcionalidades añadidas. El diagrama de bloques del diseño final es similar, y se ha mantenido el mismo conjunto de instrucciones salvo por una nueva operación, NOP u operación nula, que no tiene ninguna función salvo esperar un ciclo. El diseño final se ha implementado en el lenguaje de descripción de hardware VHDL [3].

El funcionamiento del diseño final se divide en las siguientes partes:

- **Generador de Secuencias:** Se encarga de procesar las instrucciones, ejecutando las Instrucciones de Control (administrando los niveles de llamada y los saltos de programa), y enviando las Instrucciones de Generación de Eventos al Controlador de Eventos. Dispondrá de tres bloques funcionales:

    - **Memoria de Instrucciones:** Contiene las memorias y registros asociados al programa:
        - **Memoria de Programa (PM):** Memoria RAM donde se almacenan las instrucciones de programa
        - **Contador de Programa (PC):** Contador que indica la instrucción actual a ejecutar.
        - **Registro de Instrucción (IR):** Registro que almacena la instrucción a ejecutar, previamente buscada en el ciclo anterior.

    - **Pila:** Contiene la memoria LIFO o pila donde se almacena la información correspondiente a los niveles de llamada. Se compone de:
        - **Pila (Stack):** Memoria FIFO que almacenan los niveles de llamada.
        - **Registro TOS (Top-**of-Stack): Registro que funciona como un nivel más de la pila, el superior. Sus valores pueden ser alterados externamente.

    - **Decodificador de Instrucciones:** Circuito lógico que determina el valor de las señales de control del resto de componentes del Generador de Secuencias en función de los valores del IR, el PC y el TOS.

- **Controlador de Eventos:** Se encarga de procesar las Instrucciones de Generación de Eventos, almacenándolas en una cola interna según las produce el Generador de Secuencias y ejecutándolas una a una. Se compondrá de la Cola de Eventos y el Generador de Eventos

    - **Instrucción MOV:** Se lee el patrón y el retardo indicados de sus correspondientes memorias y se guardan en sus respectivos registros. El Registro de Retardo y el Registro de Prescalado funcionan como contadores en

cascada para que el patrón se emita durante un tiempo igual al valor del retardo multiplicado por el valor del prescalado. Al llegar la cuenta a cero se lee la siguiente instrucción de la cola.

- **Instrucción SYNC:** Se comprueba cada ciclo de reloj que la entrada externa correspondiente concuerde con el valor indicado. Mientras sean distintas, se emite una señal de bloqueo STOP. Una vez las señales concuerden se desactiva la señal de bloqueo y se procede a la siguiente instrucción.

5. **Mapa de Memoria:** Actúa como un mapa de memoria convencional, dividiendo las entradas de dirección, datos y escritura entre el Generador de Secuencias y el Generador de Eventos

   Además el mapa de memoria también distingue entre las tres memorias internas contenidas en el Generador de Eventos (Memoria de Patrones, Memoria de Retardos y Registro de Prescalado).

   De esta manera, su salida es la dirección relativa a la memoria correspondiente, no la dirección absoluta en el mapa de memoria completo, y se envían solo los datos necesarios a cada parte, es decir, 16 bits al Generador de Secuencias y 32 al Generador de Eventos (para el valor por defecto de los Parámetros Genéricos).

6. **Control de Activación:** Se cuenta con mecanismos de activación y bloqueo para el Generador y el Controlador. Consideramos tres motivos para bloquear la ejecución:

   - **Bloqueo por Configuración Inicial:** El estado inicial del ASIP es el del bloqueo. Primero se cargan los datos del programa, y una vez están estos cargados, se inicia la ejecución por medio de una señal START externa.
   - **Bloqueo por Error:** En el caso de detectar cualquier error, se para la ejecución permanentemente.
   - **Bloqueo por Instrucción SYNC:** Al procesar una instrucción SYNC, se para la ejecución hasta que la instrucción se resuelva.

   Además de estos tres bloqueos generales, se considera un modo de bloqueo que sólo para la ejecución del Generador de Secuencias, el **Bloqueo por Cola Llena.** Como su nombre indica, en caso de que la Cola de Eventos esté llena se pausa la ejecución del Generador hasta que vuelva a haber espacio libre.

- **Señales de Error:** Se cuenta con 7 Señales de Salida que actúan como indicadores de errores de ejecución. Los errores contemplados son los siguientes:

  - **Dirección no Válida:** Error al escribir los datos de programa, una dirección está fuera del rango permitido.
  - **Datos no Válidos:** Error al escribir los datos de programa, un valor está fuera del rango permitido.

- **Instrucción no Válida:** Error al escribir los datos de programa, no existe instrucción para ese código.
- **Pila Rebosada:** Error al superar el número máximo de niveles de llamada en la pila.
- **Pila Vacía:** Error al intentar remover un nivel de llamada de la pila.
- **Cola Desbordada:** Error al superar el número máximo de instrucciones en cola.
- **Cola Vacía:** Error al intentar procesar la siguiente instrucción de la cola. Se debe a que hay demasiadas instrucciones no ejecutables seguidas, por lo que la ejecución no puede continuar.

4.2 Conjunto de Instrucciones

El Conjunto de Instrucciones del diseño final implementado coincide con el de los requerimientos, salvo por el añadido de una instrucción nula o de no-operación, identificada como NOP. A continuación se describirá cada instrucción en detalle en el contexto del diseño final.

Cada instrucción se compone de un OpCode de dos a cuatro bits que identifica la instrucción. El resto de bits determinan los argumentos de la instrucción. El ancho de bits de los argumentos de las instrucciones depende de varios parámetros genéricos. Todos los anchos especificados en adelante corresponden al valor por defecto de los parámetros genéricos (ej. Ancho de Instrucción de 16 bits).

Instrucciones de Generación de Eventos:

Tanto la instrucción MOV como la instrucción SYNC son mandadas a la Cola de Eventos para ser ejecutadas más tarde por el Generador de Eventos. Su funcionamiento es el siguiente:

- **MOV <IDPatrón> <IDRetardo>:**
  El OpCode que identifica esta instrucción es 00, seguida de los argumentos "IDPatrón" y "IDRetardo" cuyo ancho es de 8 y 6 bits respectivamente.

  Esta instrucción es la instrucción básica del ASIP, se usa para escribir en la salida. El Controlador de Eventos lee el Patrón y Retardos determinados por los argumentos de las memorias correspondientes y los guarda en sus respectivos registros. La señal se mantiene durante un número de ciclos de reloj igual al valor del retardo multiplicado por el valor del prescalado.

- **SYNC <Input> <Valor>:**
  El OpCode que identifica esta instrucción es 0110, seguida de los argumentos "Input" y "Valor" cuyo ancho es de 4 y 1 bits respectivamente. El resto de bits son ignorados.

  Esta instrucción es la instrucción de control del ASIP y se usa para pausar la ejecución en función de una entrada externa. La ejecución se reanuda en caso de que la entrada determinada por el valor de Input (de las 16 totales) sea igual a Valor.

  En este estado de pausado se mantienen todos los contadores del Controlador de Eventos a cero, por lo que la duración mínima de esta instrucción, en el caso de las entradas externas teniendo el valor correcto desde el principio, es de un ciclo de reloj. De la misma manera, el retardo entre la escritura del valor correcto en las entradas externas y la reanudación de la ejecución es siempre de un ciclo de reloj, en el que se descarta la instrucción SYNC y se procede a la siguiente.

Instrucciones de Control:

Ninguna de las Instrucciones de Control se manda a la Cola de Eventos, son directamente ejecutadas por el Generador de Secuencias.

- **LOOP <Count>:**
  El OpCode que identifica esta instrucción es 0101, seguida de un argumento "Count" cuyo ancho es de 12 bits.
  Esta instrucción se usa para crear bucles, repitiendo todas las instrucciones dentro del bucle un número de veces igual al valor de Count.

  Esta instrucción crea un nuevo nivel de llamada:
  - **La dirección de retorno** es la dirección actual del PC. Teniendo en cuenta el retardo de un ciclo entre leer una instrucción y ejecutarla debido al "Delay Slot", es la instrucción siguiente a ésta, es decir, la primera dentro del bucle.
  - **El número de iteraciones** es igual a Count – 1. Esto se debe a que el contador del TOS también tiene en cuenta el valor 0, por lo que el número de veces que se recorre el bucle será Count.

- **LDC <Count>**
  El OpCode que identifica esta instrucción es 0100, seguida de un argumento "Count" cuyo ancho es de 12 bits.
  Esta instrucción se usa para cambiar el número de iteraciones del nivel de llamada actual, que pasarán a ser Count – 1.

  Esta instrucción está pensada para ser usada en conjunto con la instrucción CALL, como se describe más adelante.

- **CALL <Addr>**
  El OpCode que identifica esta instrucción es 110, seguida de un argumento "Addr" cuyo ancho es de 13 bits.
  Esta instrucción se usa para llamar a una subrutina o función del programa cuya dirección de inicio es Addr.

  Esta instrucción crea un nuevo nivel de llamada:
  - **La dirección de retorno** es la dirección actual del PC + 1. Teniendo en cuenta el retardo de un ciclo entre leer una instrucción y ejecutarla debido al "Delay Slot", la dirección es dos instrucciones después de ésta.
  - **El número de iteraciones** será igual a 0, por lo que la subrutina se ejecutará una sola vez.

  Esta instrucción está pensada para usar en conjunto con la instrucción LDC, pues debido a falta de espacio en la instrucción CALL, no hay un argumento para especificar el número de iteraciones deseado.

De esta manera si se ejecuta la instrucción CALL y la siguiente es la instrucción LDC, ésta última estará ya en el "Delay Slot". De ésta manera mientras se lee la instrucción siguiente (la primera instrucción de la subrutina a la que llame CALL), se ejecutará la instrucción LDC, y en el siguiente ciclo comenzará la ejecución de la subrutina con el número de iteraciones deseado en el nivel de llamada actual.
Si no se desea especificar número de iteraciones (1 por defecto), se debe añadir una instrucción NOP en lugar de la instrucción LDC.

Este mecanismo nos permite hacer uso del "Delay Slot" para ejecutar las dos instrucciones antes de empezar la subrutina.

- **JUMP <Addr>**
  El OpCode que identifica esta instrucción es 111, seguida de un argumento "Addr" cuyo ancho es de 13 bits.
  Esta instrucción se usa para hacer un salto incondicional a otra posición de la memoria de programa, la dirección determinada por Addr. Esta instrucción no crea ningún nivel de llamada.

  Debido al salto, la instrucción siguiente a ejecutar no es la siguiente dirección en la memoria de instrucciones, sino la determinada por Addr. Por lo tanto la instrucción que se ha leído previamente, es decir el "Delay Slot", debe ser descartada.
  Para lograr esto se sobrescribe el IR con la instrucción NOP, por lo que se pierde un ciclo de reloj.

- **DEC**
  El OpCode que identifica esta instrucción es 100 y no tiene argumentos.
  Esta instrucción se usa para marcar el final de un bucle, por lo que está pensada para usar en conjunto con la instrucción LOOP.

  Esta instrucción disminuye en uno el número de iteraciones del nivel de llamada actual, enviando la señal correspondiente a la Pila. Tiene dos funciones dependiendo del número de iteraciones restantes:

  - **Cero:** Si no quedan iteraciones pendientes, no se realiza ninguna función adicional, y la Pila eliminará el nivel de llamada actual automáticamente, por lo que la siguiente instrucción que se ejecuta es la inmediatamente posterior a DEC, es decir, la primera después del bucle.

  - **Más de Cero:** Si quedan iteraciones pendientes, se deberá volver al principio del bucle, por lo que se salta a la dirección de retorno del nivel de llamada actual. Como se ha realizado un salto, la instrucción actual del "Delay Slot" no es válida, y se debe descartar sustituyéndola con una instrucción NOP, perdiendo un ciclo de reloj en el proceso.

- **DECRET <Addr>**

El OpCode que identifica esta instrucción es 101, seguida de un argumento “Addr” cuyo ancho es de 13 bits.
Esta instrucción se usa para marcar el final de una subrutina, por lo que está pensada para usar en conjunto con la instrucción CALL. Addr debe ser la dirección de comienzo de la subrutina.

Esta instrucción disminuye en uno el número de iteraciones del nivel de llamada actual, enviando la señal correspondiente a la Pila. Tiene dos funciones dependiendo del número de iteraciones restantes:

- **Cero:** Si no quedan iteraciones pendientes se debe volver a la instrucción siguiente a la llamada a subrutina, es decir, se debe saltar a la dirección de retorno del nivel de llamada. La Pila elimina el nivel de llamada automáticamente.

- **Más de Cero:** Si quedan iteraciones pendientes, se debe volver al principio de la subrutina, por lo que se salta a la dirección especificada por Addr.

En ambos casos, como se ha realizado un salto la instrucción actual del “Delay Slot” no es válida, y se debe descartar sustituyéndola con una instrucción NOP, perdiendo un ciclo de reloj en el proceso.

- **NOP**
  El OpCode que identifica esta instrucción es 0111. Esta instrucción es una instrucción nula que no realiza ninguna operación.

4.3 Arquitectura

4.3.1 Interfaz del Sistema

Fichero VHDL Global: ASIP.vhd

Nombre de la Entidad: ASIP.

La Interfaz de conexión del diseño con otros sistemas externos como su implementación en una FPGA o su empaquetado como IP (Intelectual Property) para uso en futuros diseños cuenta con los siguientes puertos y parámetros de personalización:

Parámetros Configurables

Los parámetros genéricos configurables del diseño final, su correspondencia con los parámetros genéricos de la entidad VHDL ASIP, así como su valor por defecto se observan en la Tabla 6.

Parámetro del Diseño	Parámetro de la Entidad	Valor por Defecto	Descripción
NPatrones	NPatterns	256	Número de direcciones de memoria en la Memoria de Patrones
NSalidas	NOutputSignals	32	Ancho de bits de los patrones y la salida de datos
NRetardos	NDelays	64	Número de direcciones de memoria en la Memoria de Retardos
AnchoRetardo	DelayWidth	20	Ancho de bits de los retardos
NInstrucciones	NInstructions	8192	Número de direcciones de memoria en la Memoria de Instrucciones
AnchoInstrucción	InstructionWidth	16	Ancho de bits de las instrucciones
TamañoPila	StackSize	8	Número de direcciones de memoria de la Pila (sin contar el Registro TOS)
MaxIteraciones	MaxIterations	4096	Valor máximo que puede tomar el número de iteraciones de un nivel de llamada.
NEntradas	NInputSignals	16	Ancho de bits de la entrada externa.
MaxPrescalado	MaxPrescale	10	Valor máximo del prescalado
TamañoCola	QueueSize	16	Número de direcciones de la Cola de Eventos

Tabla 6: Parámetros Genéricos Configurables del Diseño Final Implementado

Puertos de Entrada y Salida

Los puertos de entrada y salida del diseño final se observan en la Tabla 7.

Puerto	I/O	Tipo	Ancho	Descripción
Clk	Entrada	Reloj	1 bit	Señal de Reloj
Reset	Entrada	Reinicio	1 bit	Señal de Reinicio Asíncrona
WE	Entrada	Señal	1 bit	Señal de Escritura en Memoria
Address	Entrada	Bus	14 bits*	Bus de Dirección de Escritura
Data	Entrada	Bus	32 bits*	Bus de Datos de Escritura
Input	Entrada	Señal	16 bits*	Conjunto de 16 entradas independientes externas.
Output	Salida	Bus	32 bits*	Bus de Salida de Datos
Error	Salida	Indicador (Flag)	7 bits	Conjunto de 7 señales indicadoras de error.

Tabla 7: Puertos de Entrada y Salida del Diseño Final Implementado

Anchos variables:

El ancho de los puertos marcados con * depende de los parámetros genéricos, siendo los valores de la tabla los valores por defecto.

1. **Address:** Su ancho es igual a 2 * $\log_2 NInstrucciones$, de esta manera abarca el doble de la memoria de instrucciones, siendo la segunda mitad el mapeo de memoria de las memorias restantes.
2. **Data y Output:** Su ancho es igual a NSalidas.
3. **Input:** Su ancho es igual a NEntradas.

4.3.2 Diagrama de bloques básico

El diagrama de bloques final del diseño se observa en la Figura 3.

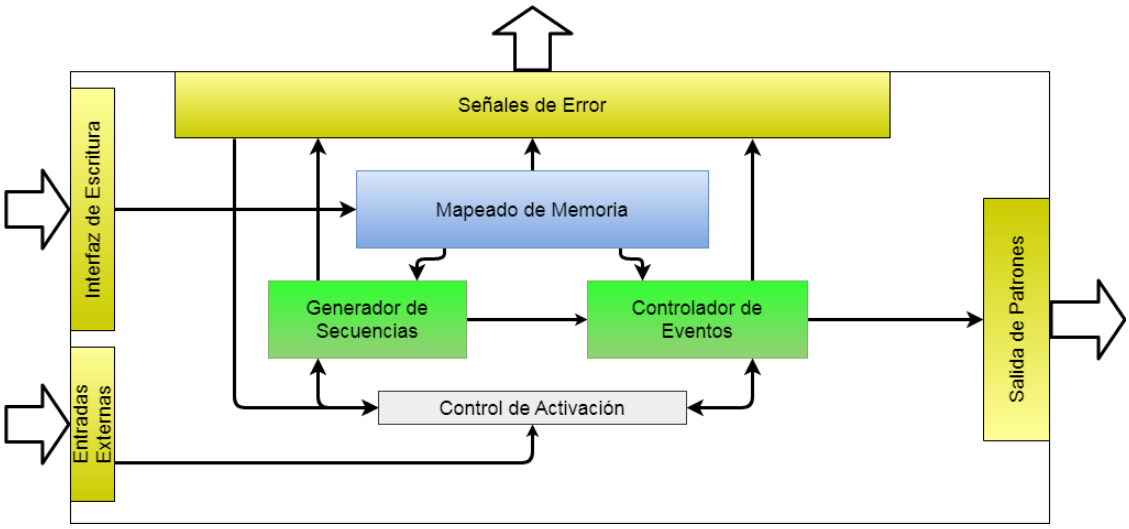

Figura 3: Diagrama de Bloques general del Diseño Final Implementado

Observamos cuatro bloques lógicos, así como 4 puertos de entrada y salida en naranja.

Bloques de Lógica:

El ASIP está dividido en tres bloques principales, siendo sus conexiones las siguientes:

- **Mapeado de Memoria:**
  La función de este bloque es la de crear un mapa de memoria a partir de las Memorias Internas individuales
  Este bloque recibe las entradas de la interfaz de escritura (Address, Data y WE), y las divide entre los otros dos bloques lógicos:

  - **Salidas al Generador de Secuencias:** Address, Data y WE.
  - **Salidas al Controlador de Eventos:** Una señal de Datos común y un par de señales de Escritura y Dirección por cada una de sus tres memorias internas (patrones, retardos y prescalado), salvo la señal de dirección de del Prescalado, pues al ser un único registro solo ocupa una dirección de memoria. En total son 6 salidas.

  El Mapa también cuenta con dos Salidas de Errores, Error de Dirección no Válida y Error de Datos no Válidos.

- **Generador de Secuencias:**
  La función de este bloque es la de leer procesar las instrucciones, ejecutando las Instrucciones de Control y enviado las Instrucciones de Generación de Eventos al Controlador de Eventos.
  Este bloque tiene las siguientes conexiones con otros bloques:

- **Entradas del Mapeado de Memoria:** Son las señales de escritura de datos en la memoria de programa que se usan previamente a la ejecución.
- **Entrada y Salida al Control de Activación:** El Generador recibe una señal de Activación (Enable) del Control de Activación, y envía una señal de Instrucción Ejecutable que se usa para determinar el momento de puesta en marcha del Controlador.
- **Salida al Controlador de Eventos:** El Generador envía la instrucción actual al Controlador junto con una señal (Enqueue) que determina si la instrucción se debe introducir en la cola o ignorar.
- **Salidas de Errores:** El Generador cuenta con tres salidas de errores:
  - **Overflow:** Rebosado de la pila.
  - **Underflow:** Error al intentar extraer un registro de la pila vacía.
  - **Instrucción no Válida:** Código de Instrucción no definido.
- Este bloque es **Secuencial** por lo que cuenta con una entrada de Reloj y otra de Reinicio.

- **Controlador de Eventos:**
  La función de este bloque es la de ejecutar las Instrucciones de Generación de Eventos provenientes del Generador de Secuencias.
  Este bloque tiene las siguientes conexiones con otros bloques:

  - **Entradas del Mapeado de Memoria:** Son las señales de escritura de datos en las Memorias de Retardos y Patrones y el Registro de Prescalado que se usan previamente a la ejecución.
  - **Entrada y Salida al Control de Activación:** El Controlador recibe una señal de Activación (Enable) del Control de Activación, y envía una señal de Cola Llena que se usará para determinar si se debe pausar el Generador de Secuencias.
  - **Entrada del Generador de Secuencias:** El Controlador recibe la instrucción actual del Generador junto con una señal (Instrucción Ejecutable) que determina si la instrucción se deberá introducir en la cola o ignorar.
  - **Salidas de Errores:** El Controlador cuenta con dos salidas de errores:
    - **Overflow:** Rebosado de la Cola de Eventos.
    - **Underflow:** Error al intentar extraer un registro de la cola vacía.
  - Este bloque es **Secuencial** por lo que cuenta con una entrada de Reloj y otra de Reinicio.

Control de Activación:

Además de los bloques descritos anteriormente, el ASIP cuenta con una lógica de Control de Activación que regula el pausado y reanudación de la ejecución del Generador de Secuencias y el Controlador de Eventos. Cuenta con dos registros de un bit o flags llamados flags START, que indican si su respectivo bloque ha empezado la ejecución o no.

4.3.3 Mapeado de Memoria

Nombre del Archivo: MemoryMapper.vhd

Nombre de la Entidad: MemoryMapper.

Interfaz

Parámetros Genéricos:

- **Ninstructions:** Tamaño de la Memoria de Instrucciones.
- **InstructionWidth:** Ancho de Bits de las Instrucciones.
- **NPatterns:** Tamaño de la Memoria de Patrones.
- **NOutputSignals:** Ancho de Bits de los Patrones.
- **NDelays:** Tamaño de la Memoria de Retardos.
- **DelayWidth:** Ancho de Bits de los Retardos.
- **MaxPrescale:** Valor Máximo del Prescalado.

Puertos:

- **WE (Entrada):** Señal de Escritura.
- **Address(Entrada):** Señal de Dirección de Escritura.
- **Data(Entrada):** Datos de Escritura.
- **WE_Instruction (Salida):** Señal de Escritura a la Memoria de Instrucciones.
- **WE_DelayTable (Salida):** Señal de Escritura a la Memoria de Retardos.
- **WE_PatternTable (Salida):** Señal de Escritura a la Memoria de Patrones.
- **Load_Prescale (Salida):** Señal de Escritura al Registro de Prescalado.
- **Address_Instruction (Salida):** Dirección a la Memoria de Instrucciones.
- **Address_PatternTable (Salida):** Dirección a la Memoria de Patrones.
- **Address_DelayTable (Salida):** Dirección a la Memoria de Retardos.
- **Invalid_Address (Salida):** Señal de Error de Dirección no Válida.
- **Invalid_Data (Salida):** Señal de Error de Datos no Válidos.

Descripción Técnica

El Mapa de Memoria genera señales de Escritura de Datos para cada una de las tres memorias internas del ASIP además del Registro de Prescalado en función de la Dirección de Memoria indicada por la señal Address.

Cuando Address indique una de las Memorias Internas, las señales de salida a las otras memorias tomarán el valor 0. La Señal de Datos de la memoria objetivo es capada al Ancho de Bits de la memoria en cuestión, justificado a la derecha (es decir, sólo se transmitirán los bits menos significativos). La señal de dirección de salida es el valor decimal basado en cero relativo a la memoria a la que se está escribiendo. (Ej. Si Address indica la tercera dirección de la Memoria de Patrones, Address_PatternTable valdrá 2.)

La división se produce de la siguiente manera, en función de Address, y en función de los parámetros genéricos del diseño, como se observa en la Tabla 8. Para los valores por defecto de los parámetros genéricos, la división se describe en la Tabla 9.

<table>
<tr><th colspan="2">Dirección</th><th rowspan="2">Memoria Objetivo</th><th rowspan="2">Salida de Dirección</th><th rowspan="2">Salida de Datos</th></tr>
<tr><th>Inicial</th><th>Final</th></tr>
<tr><td>0</td><td>NInstrucciones - 1</td><td>Memoria de Instrucciones</td><td rowspan="3">Address menos Dirección Inicial de esta sección de memoria.</td><td>Bits menos significativos de Data, de ancho AnchoInstrucción</td></tr>
<tr><td>NInstrucciones</td><td>NInstrucciones + NPatrones - 1</td><td>Memoria de Patrones</td><td>Entrad de datos completa</td></tr>
<tr><td>NInstrucciones + NPatrones</td><td>NInstrucciones + NPatrones + NRetardos - 1</td><td>Memoria de Retardos</td><td>Bits menos significativos de la Data, de ancho AnchoRetardo</td></tr>
<tr><td colspan="2">NInstrucciones + NPatrones + NRetardos</td><td>Registro de Prescalado</td><td>Ninguna</td><td>Bits menos significativos de Data, de ancho $\log_2(MaxPrescalado)$</td></tr>
<tr><td>NInstrucciones + NPatrones + NRetardos + 1</td><td>(2 * Instrucciones) - 1</td><td colspan="3">Dirección no Válida (Se activa Invalid_Address)</td></tr>
</table>

Tabla 8: Mapeado de Memoria

Dirección		Memoria	Salida de Dirección	Ancho de Salida de Datos
Inicial	Final			
0x0000	**0x1FFF**	Generador de Secuencias	Address	16 bits menos significativos de Data
0x2000	**0X20FF**	Memoria de Patrones	Address menos 0x2000	32 bits completos de Data
0x2100	**0X213F**	Memoria de Retardos	Address menos 0x2100	20 bits menos significativos de Data

Dirección		Memoria	Salida de Dirección	Ancho de Salida de Datos
Inicial	Final			
0x2140		Registro de Prescalado	Ninguna	4 bits menos significativos de Data
0x2141	**0X3FFF**	Dirección no Válida (Se activa Invalid_Address)		

Tabla 9: Mapeado de Memoria para valores por defecto de los Parámetros Genéricos

La señal Invalid_Data se activará cuando Data no sea compatible con la salida de datos descrita en la tabla. Esto se debe a que el valor de Data está fuera del rango de datos válidos para cada memoria. Los rangos de datos validos son de 0 al máximo posible para el Ancho de Salida de Datos para las tres memorias, y de 1 a MaxPrescale para el Registro de Prescalado.

Observaciones

Los Parámetros Genéricos se heredan de la entidad superior, ASIP, por lo que si se configuran mal éstos de acuerdo con las observaciones del apartado 4.4, el Mapeo de Memoria no funcionará correctamente.

Las Señales de Error dependen únicamente de los datos de escritura, por lo que una activación de estos errores indica un problema en el programa cargado en el ASIP.

4.3.4 Generador de Secuencias

Nombre del Archivo: SequenceGenerator.vhd

Nombre de la Entidad: SequenceGenerator.

Interfaz

Parámetros Genéricos:

- **Ninstructions:** Tamaño de la Memoria de Instrucciones.
- **InstructionWidth:** Ancho de Bits de las Instrucciones.
- **StackSize:** Tamaño de la Pila.
- **MaxIterations:** Número de Iteraciones Máximas de un Nivel de Llamada.

Puertos:

- **Clk (Entrada):** Señal de Reloj.
- **Reset (Entrada):** Señal de Reinicio Síncrono.
- **WE (Entrada):** Señal de Escritura.
- **Enable (Entrada):** Señal de Activación.
- **Address (Entrada):** Señal de Dirección de Escritura.
- **Data (Entrada):** Señal de Datos de Escritura.
- **Instruction (Salida):** Instrucción actual siendo procesada.
- **Enqueue (Salida):** Señal de Puesta en Cola.
- **Overflow (Salida):** Señal de Error de Pila Rebosada.
- **Underflow (Salida):** Señal de Error de Pila Vacía.
- **Invalid_Instruction (Salida):** Señal de Error de Instrucción no Válida.

Su función será la de administrar los niveles de llamada y enviar las Instrucciones de Generación de Eventos a la Cola de Eventos. El diagrama de bloques del Generador se observa en la Figura 4.

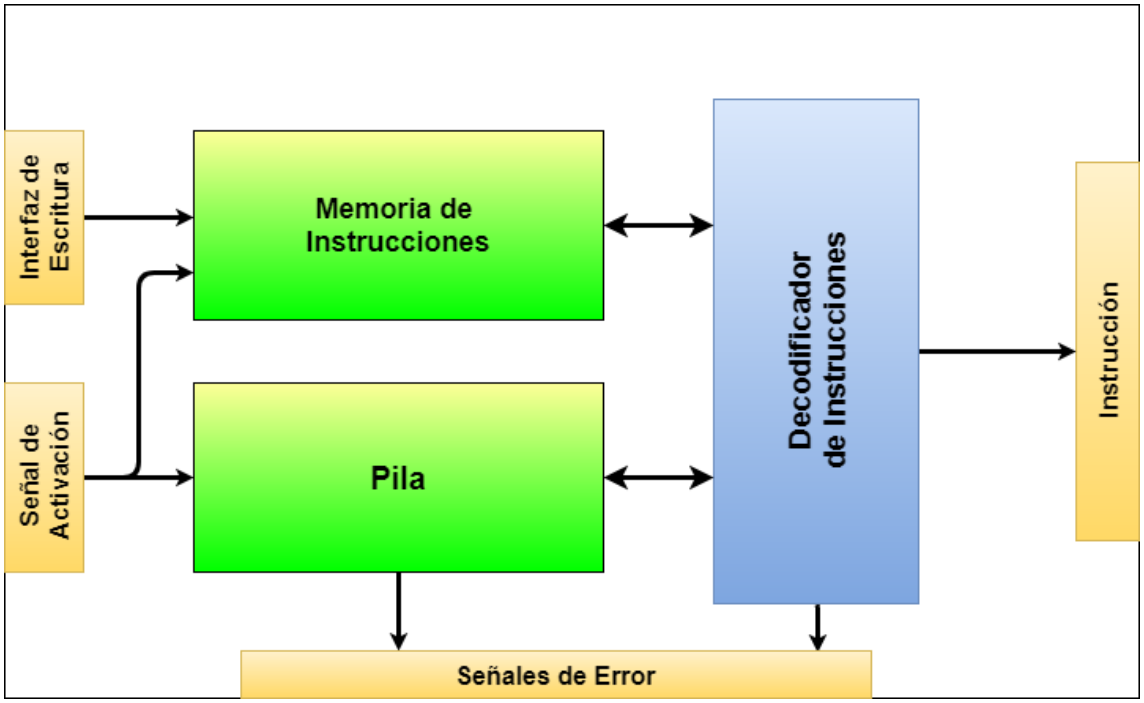

Figura 4: Diagrama de Bloques del Generador de Secuencias

El funcionamiento del Generador es el siguiente:

Se procesan las instrucciones leídas de la Memoria de Instrucciones mediante el Decodificador, que determinara el estado de las señales de control del resto de componentes en función del tipo de instrucción que se esté procesando en ese ciclo, cambiando el valor del Contador de Programa, creando y descartando niveles de llamada según la instrucción lo requiera. Se detallará el funcionamiento de cada bloque más adelante.

El procesado de las instrucciones está segmentado o en "pipeline", es decir, dividido en etapas:

- **Etapa de búsqueda o "Delay Slot":** En esta etapa se lee la instrucción de la Memoria de Programa en la dirección indicada por el contador de programa. La instrucción se almacena en un registro intermedio, el Registro de Instrucción (IR).

- **Etapa de ejecución:** En esta etapa el Decodificador de Instrucciones ejecuta la instrucción almacenada en el IR.

De esta manera al ejecutar una instrucción se busca la siguiente. Debido a esto hay que tener en cuenta que algunas Instrucciones de Control provocan que la instrucción del Delay Slot deba ser descartada, pues la siguiente instrucción a ejecutar no es la inmediatamente posterior. En este caso se descarta el Delay Slot, y esa instrucción se debe ignorar. Este mecanismo se ha implementado mediante una nueva instrucción, NOP, que no realiza ninguna operación. Para descartar el Delay Slot se sobrescribirá el IR con la instrucción NOP en vez de leer la instrucción de la Memoria. Debido a esto se pierde un ciclo de reloj.

NOP también es el valor inicial de IR al comienzo de la ejecución, pues se debe esperar a leer la primera instrucción antes de empezar a ejecutar.

Podemos observar este funcionamiento de manera más simple con un ejemplo, en la Tabla 10.

Contador de Programa	Etapa de Búsqueda (Delay Slot)	Etapa de Ejecución
0	MOV 1,3	NOP
1	LOOP 3	MOV 1,3
2	MOV 3, 4	LOOP 3
3	JUMP 0	MOV 3, 4
4	**MOV 0,0**	JUMP 0
0	MOV 1,3	**NOP**
1	LOOP 3	MOV 1,3

Tabla 10: Ejemplo de Flujo de Ejecución

Podemos observar en la Tabla 10 como la instrucción 3 es una instrucción que provoca un salto o “branching”, luego en el siguiente ciclo en vez de ejecutarse la instrucción guardada en el Delay Slot (instrucción 4) se ejecuta una instrucción NOP.

Memoria de Instrucciones

Nombre del Archivo: Instruction_Memory.vhd

Nombre de la Entidad: Instruction_Memory.

Memoria de Instrucciones: Interfaz

Parámetros Genéricos:

- **Ninstructions:** Tamaño de la Memoria de Instrucciones.
- **InstructionWidth:** Ancho de Bits de las Instrucciones.

Puertos:

- **Clk (Entrada):** Señal de Reloj.
- **WE (Entrada):** Señal de Escritura.
- **Load_PC (Entrada):** Señal de Escritura del Contador de Programa.
- **Reset (Entrada):** Señal de Reinicio Síncrono.
- **Enable (Entrada):** Señal de Activación.
- **Address(Entrada):** Dirección de Escritura.
- **PC_In (Entrada):** Datos de Escritura al Contador de Programa.
- **Instruction_In (Entrada):** Datos de Escritura a la Memoria de Instrucciones.
- **NOP (Entrada):** Señal de Descarte del Delay Slot.
- **Instruction_Out (Salida):** Instrucción a procesar.
- **PC_Out (Salida):** Valor actual del Contador de Programa (PC).

Memoria de Instrucciones: Diseño Técnico

Es el conjunto del Contador de Programa (PC), la Memoria de Programa (PM) y el Registro de Instrucción (IR). Su diagrama de bloques se observa en la Figura 5.

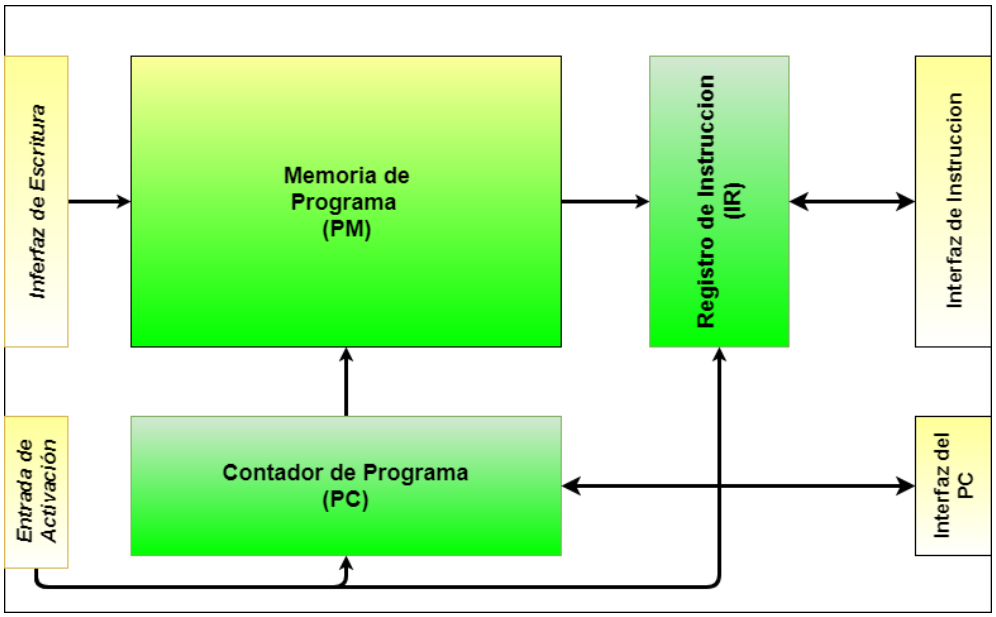

Figura 5: Diagrama de Bloques de la Memoria de Instrucciones

El PC determina que dirección de la PM se lee, y ésta se almacena en el IR cada ciclo de reloj. El PC se incrementa en uno cada ciclo de reloj y se puede cargar con un valor arbitrario desde la entrada apropiada.

Se puede sobrescribir el IR con la operación NOP (operación nula) en el caso de que haya que descartar la instrucción pre-leída debido a un salto o "branching" en el programa.

La PM y el PC serán modelados como una "RAM Genérica" y un "Contador Genérico", dos componentes de uso general que se describen más adelante, en el apartado 4.3.6

Memoria de Instrucciones: Observaciones

Los Parámetros Genéricos se heredan de la entidad superior, Generador de Secuencias, por lo que si se configuran mal éstos de acuerdo con las observaciones del apartado 4.4, la Memoria de Instrucciones no funcionará correctamente.

El programa debe contar siempre con un bucle infinito. Si no se cumple esta regla, al procesar la última instrucción del programa escrito, la siguiente instrucción se leerá de un registro de la memoria que contiene basura, por lo que se ejecutará una instrucción no deseada o se producirá un error de Instrucción no Válida.

El número máximo de instrucciones que se pueden almacenar en la memoria es NInstrucciones. Si el programa continua pasadas éstas, el PC vuelve al valor 0 y el programa vuelve a empezar.

Pila

Nombre del Archivo: Stack_Block.vhd

Nombre de la Entidad: Stack_Block.

Pila: Interfaz

Parámetros Genéricos:

- **StackSize:** Tamaño de la Pila
- **Ninstructions:** Tamaño de la Memoria de Instrucciones.
- **MaxIterations:** Número máximo de iteraciones por Nivel de Llamada

Puertos:

- **Clk (Entrada):** Señal de Reloj.
- **Reset (Entrada):** Señal de Reinicio Síncrono.
- **Enable (Entrada):** Señal de Activación.
- **Push (Entrada):** Señal de Operación PUSH.
- **Load_TOS_Count (Entrada):** Señal de Escritura del Número de Iteraciones del Registro TOS.
- **Dec_TOS (Entrada):** Señal de Decremento del Número de Iteraciones del Registro TOS.
- **Addr_In (Entrada):** Nueva Dirección de Retorno.
- **Count_In (Entrada):** Nuevo Número de Iteraciones.
- **Addr_Out (Salida): D**irección de Retorno actual del Registro TOS.
- **Count_Out (Salida): N**úmero de Iteraciones actual del Registro TOS.
- **Overflow (Salida):** Señal de Error de Pila Rebosada.
- **Underflow (Salida):** Señal de Error de Pila Vacía.

Es un bloque que actúa como una memoria LIFO o pila, compuesto por una memoria LIFO y un registro que actúa como parte superior de la pila al que llamaremos registro TOS (Top-Of-Stack). Así como un sistema de control y una serie de indicadores o Flags. Esto permite alterar este registro desde fuera de la pila. Su diagrama de bloques se observa en la Figura 6.

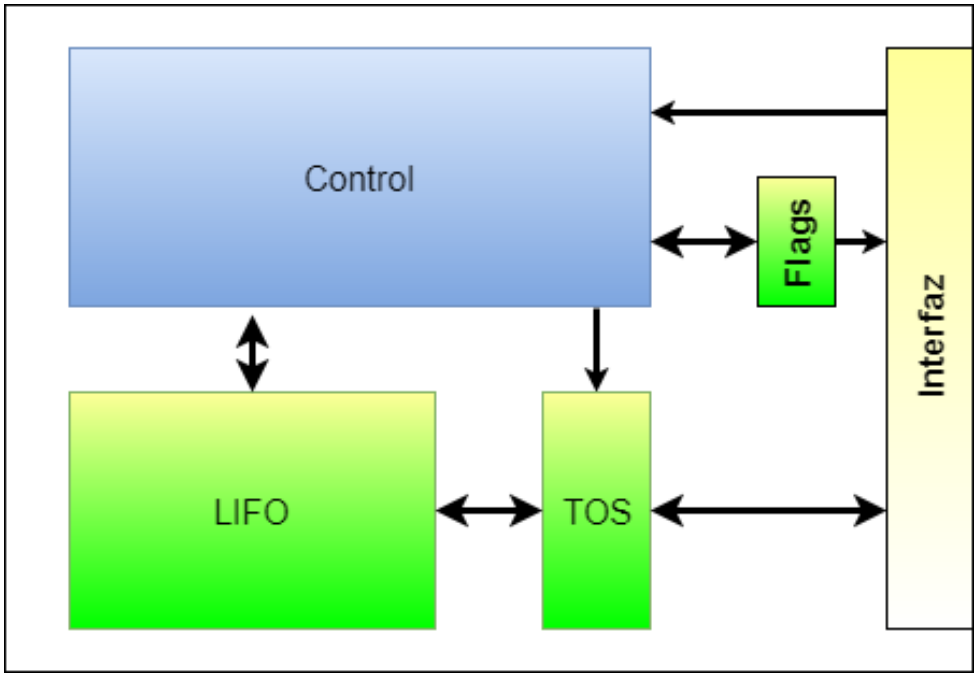

Figura 6: Diagrama de Bloques de la Pila

Cada registro de la LIFO, así como el TOS se compone de dos partes

- Un **valor de dirección** que indica la dirección de retorno: al finalizar una iteración en el caso de los bucles, y al finalizar el nivel de llamada en el caso de llamadas a subrutinas.
- Un **número de iteraciones** restantes del nivel de llamada actual.

La funcionalidad del bloque es la siguiente:

- **LIFO y TOS:**
  En conjunto funcionan como una pila o memoria LIFO, siendo TOS un registro superior añadido al tope de la memoria LIFO interna. Así pues cuando se introduce un nuevo registro en el bloque, este se introduce al TOS, empujando su valor previo a la LIFO. De la misma manera, al extraer un valor, se extrae el valor del TOS y el valor superior de la LIFO pasa al TOS.

  Además de este funcionamiento, el valor del TOS puede ser escrito y leído externamente, y su número de iteraciones puede ser disminuido en uno.

- **Flags:**
  El bloque con cuatro indicadores o flags, de los cuales dos son operacionales y los otros dos son señales de error. Son los siguientes:
  - **Flags de Pila Vacía y Llena:** Indican si el conjunto LIFO-TOS está vacío o lleno de datos respectivamente. Son usados por el Control.
  - **Flags de Overflow y Underflow:** Señales de error que indican un desborde de la pila (Overflow) o que se ha intentado extraer un valor con la pila vacía (Underflow).

- **Control:**
  El control administra tres tipos de operaciones distintas:

  - **Escritura del TOS:**
    Una vez escrita, la dirección de retorno del TOS es de solo lectura, pero se puede sobrescribir el número de iteraciones, tanto con un valor concreto o disminuyendo en uno el valor previo.

  - **PUSH:**
    Se introduce un nuevo nivel de llamada, escribiendo sus datos en el TOS.

    En caso de que el Flag de Pila Llena esté inactivo, se comprueba el Flag de Pila Vacía y, en caso de que esté inactivo, se mueven los datos previos del TOS a la LIFO. Se desactiva el Flag de Pila Vacía.
    Si al realizar esta operación se llena la LIFO, se activa el Flag de Pila Llena.

    En el caso de que el Flag de Pila Llena estuviese activo, no se realiza ninguna operación, sino que se activa el Flag de Overflow.

  - **POP:**
    Se extrae el nivel de llamada superior, descartando los datos actuales del TOS.

    En caso de que el Flag de Pila Vacía esté inactivo, se mueve el registro superior de la LIFO al TOS, o en caso de que la LIFO esté vacía, se activa el Flag de Pila Vacía. Se desactiva el Flag de Pila Llena.

    En caso de que el Flag de Pila Vacía esté activo, no se realiza ninguna operación, sino que se activa el Flag de Underflow.

    Esta operación no se activa mediante una entrada externa sino que es automática. Se ejecuta cuando se va a disminuir en uno el número de iteraciones del nivel de llamada actual y éste es cero.

Pila: Memoria LIFO

Nombre del Archivo: LIFO.vhd

Nombre de la Entidad: LIFO.

Parámetros Genéricos:

- **StackSize:** Tamaño de la Pila
- **Ninstructions:** Tamaño de la Memoria de Instrucciones.
- **MaxIterations:** Número máximo de iteraciones por Nivel de Llamada

Puertos:

- **Clk (Entrada):** Señal de Reloj.
- **Reset (Entrada):** Señal de Reinicio Síncrono.
  Push (Entrada): Señal de Operación PUSH.
- **Pop (Entrada):** Señal de Operación POP.
- **Addr_in (Entrada):** Nueva dirección de retorno.
- **Count_in (Entrada):** Nuevo número de iteraciones.
- **Addr_out (Salida):** Dirección de retorno actual del tope de la pila.
- **Count_out (Salida): N**úmero de iteraciones actual del tope de la pila.
- **Full (Salida):** Señal de Pila Llena.
- **Empty (Salida):** Señal de Pila Vacía.
- **Overflow (Salida):** Señal de Error de Pila Rebosada.
- **Underflow (Salida):** Señal de Error de Pila Vacía.

Funcionamiento:

Funciona como una memoria LIFO tradicional, cuyo número de direcciones de memoria es StackSize. Cada registro está compuesto de dos valores distintos, un valor representa el Número de Iteraciones del Nivel de Llamada y el otro su Dirección de Retorno. El tamaño de estos registros, y por tanto el tamaño de la pila, está determinado por MaxIteraciones para el Número de Iteraciones y NInstrucciones para la dirección de retorno.

Pila: Observaciones

Los Parámetros Genéricos se heredan de la entidad superior, Generador de Secuencias, por lo que si se configuran mal éstos de acuerdo con las observaciones del apartado 4.4, la Pila no funcionará correctamente.

El número máximo de Niveles de Llamada es igual al tamaño de la Pila más uno, debido al registro TOS (Por defecto, un total de 9). Los programas del ASIP se deben crear teniendo esto en cuenta para no provocar un error de Pila Rebosada.

Decodificador de Instrucciones

Nombre del Archivo: Instruction_Decoder.vhd

Nombre de la Entidad: Instruction_Decoder.

Decodificador de Instrucciones: Interfaz

Parámetros Genéricos:

- **InstructionWidth:** Ancho de Bits de las Instrucciones.
- **Ninstructions:** Tamaño de la Memoria de Instrucciones.
- **MaxIterations:** Número máximo de iteraciones por Nivel de Llamada

Puertos:

- **Instruction_In (Entrada):** Instrucción actual a procesar.
- **TOS_In_addr (Entrada): D**irección de Retorno actual del Registro TOS.
- **TOS_In_count (Entrada): N**úmero de Iteraciones actual del Registro TOS.
- **PC_In (Entrada):** Dirección actual del Contador de Programa.
- **Enqueue (Entrada):** Señal de Puesta en Cola
- **Load_TOS_Count (Salida):** Señal de Escritura del Número de Iteraciones del Registro TOS.
- **Push (Salida):** Señal de Operación PUSH.
- **Dec_TOS (Salida):** Señal de Decremento del Número de Iteraciones del Registro TOS.
- **Load_PC (Salida):** Señal de Escritura del Contador de Programa
- **TOS_Out_addr (Salida):** Nueva Dirección de Retorno.
- **TOS_Out_count (Salida):** Nuevo Número de Iteraciones.
- **NOP (Salida):** Señal de descarte del Delay Slot.
- **NDEF (Salida):** Señal de Error de Instrucción no Válida.
- **PC_Out (Salida):** Nueva Dirección del Contador de Programa

Decodificador de Instrucciones: Descripción Técnica

Es el bloque lógico que interpreta las señales leídas de los otros dos bloques y calcula las señales de control correspondientes, así como envía las Instrucciones de Generación de Eventos a la Cola. El valor de estas salidas está determinado por el tipo de instrucción que se lea. El funcionamiento de cada tipo de instrucción ha sido explicado previamente en el apartado 4.2.

Los datos que recibe son las siguientes:

- **Instrucción (Instruction_In):** Leída de la Memoria de Instrucciones
- **PC (PC_In):** Valor actual del contador de programa, leída de la Memoria de Instrucciones
- **TOS (TOS_In_addr y TOS_In_Count):** Valor actual del registro TOS leído de la pila. Incluye dirección de retorno y número de iteraciones.

Las señales de control que emite son las siguientes:

- **Señal de Puesta en Cola (Enqueue):** Señal que marca la instrucción actual como Instrucción de Generación de Eventos y permite que se introduzca en la Cola de Eventos.
- **Señales del PC (PC_Out y Load_PC):** Par de señales que permiten escribir un valor arbitrario al PC
- **Señales del TOS (TOS_Out_addr, TOS_Out_count, Load_TOS_count, Dec_TOS y Push):** Señales que permiten escribir un valor arbitrario al número de iteraciones del TOS o reducirlo en uno, así como introducir un nuevo nivel de llamada (PUSH).
- **Señal de NOP (NOP):** Señal que permite cargar la operación NOP en el Registro de Instrucciones (IR) en lugar de la instrucción determinada por el PC. Esto se debe a que ciertas Instrucciones de Control provocan que la instrucción previamente buscada y almacenada en el IR no sea la siguiente a ejecutar, por lo que se introduce una instrucción nula en el ciclo de reloj perdido.

Por último, se dispone de una Señal de Error de Instrucción no Válida, NDEF, que indica que se ha intentado procesar una instrucción no definida, es decir, no se reconoce el tipo de instrucción.

A continuación se describe cada instrucción en detalle en el contexto del diseño final, incluyendo las señales que activará el Decodificador de Instrucciones para procesarlas:

- **Instrucciones de Generación de Eventos:**

  Tanto la instrucción MOV como la instrucción SYNC se mandan a la Cola de Eventos, manteniendo todas las señales de control del Decodificador inactivas, salvo la señal de encolado.

- **Instrucciones de Control:**

Ninguna de las Instrucciones de Control se manda a la Cola de Eventos. Cada instrucción se compone de un Código de Operación (OpCode) de dos a cuatro bits que identifican el tipo de instrucción. El resto de bits son usados por los argumentos de la instrucción.

El ancho de bits de los argumentos de las instrucciones depende de varios parámetros genéricos. Todos los anchos especificados en adelante se corresponden con el valor por defecto de los parámetros genéricos (ej. Ancho de Instrucción de 16 bits).

Podemos ver en la Tablas 11 y 12 un resumen de las señales de control que activa cada instrucción. En la primera tabla podemos observar las señales de control a la Memoria de Instrucciones para las distintas Instrucciones. En la Tabla 11 aparecen todas las señales de control, salvo las señales de control de la Pila, que aparecen en la Tabla 12.

Instruction_In		Enqueue	Señales del PC		NOP
			Load_PC	PC_Out	
MOV <IDPatrón> <IDRetardo>		1	0	N/A	0
SYNC <Input> <Valor>		1	0	N/A	0
LOOP <Count>		0	0	N/A	0
LDC <Count>		0	0	N/A	0
CALL <Addr>		0	1	PC_In	0
JUMP <Addr>		0	1	PC_In	1
DEC	**TOS_In_count > 0**	0	1	TOS_In_addr	1
	TOS_In_count = 0	0	0	N/A	0
DECRET <Addr>	**TOS_In_count > 0**	0	1	<Addr>	1
	TOS_In_count = 0	0	1	TOS_In_addr	1
NOP		0	0	N/A	0

Tabla 11: Decodificación de Instrucciones. Señales de la Memoria de Instrucciones.

Instruction_In	Señales de la Pila				
	Dec_TOS	Load_TOS_count	PUSH	Salida al TOS	
				TOS_Out_addr	TOS_Out_count
MOV <IDPatrón> <IDRetardo>	0	0	0	N/A	N/A
SYNC <Input> <Valor>	0	0	0	N/A	N/A
LOOP <Count>	0	0	1	PC_In	<Count>
LDC <Count>	0	1	0	N/A	<Count>
CALL <Addr>	0	0	1	PC_In + 1	1
JUMP <Addr>	0	0	0	N/A	N/A
DEC	1	0	0	N/A	Tos_In_count - 1
DECRET <Addr>	1	0	0	N/A	Tos_In_count - 1
NOP	0	0	0	N/A	N/A

Tabla 12: Decodificación de las Instrucciones. Señales de la Pila.

Decodificador de Instrucciones: Observaciones

Los Parámetros Genéricos se heredan de la entidad superior, Generador de Secuencias, por lo que si se configuran mal éstos de acuerdo con las observaciones del apartado 4.4, el Decodificador de Instrucciones no funcionará correctamente.

Como se ha descrito en el apartado 4.2, algunas instrucciones están pensadas para ser usadas en conjunto con otras, un ejemplo claro siendo la instrucción CALL que debe ser seguida de una instrucción LDC o NOP. El Decodificador asume que estas reglas se están siguiendo, y genera señales de control de acuerdo a ello. Debido a esto, un programa escrito sin tener en cuenta estas reglas puede provocar resultados no esperados o señales de error.

Observaciones

Los Parámetros Genéricos se heredan de la entidad superior, ASIP, por lo que si se configuran mal éstos de acuerdo con las observaciones del apartado 4.4, el Generador de Secuencias no funcionará correctamente.

Se debe tener en cuenta una serie de restricciones en el programa para evitar los errores de ejecución.

- No se debe superar el número máximo de niveles de llamada permitidos por los parámetros genéricos, debido a esto se debe vigilar el número de bucles y llamadas a subrutinas anidados en el programa.
- Se debe tener en cuenta las instrucciones que se usan en combinación con otras.

4.3.5 Controlador de Eventos

Nombre del Archivo: Event_Controller.vhd

Nombre de la Entidad: Event_Controller.

Interfaz

Parámetros Genéricos:

- **InstructionWidth:** Ancho de Bits de las Instrucciones.
- **NPatterns:** Tamaño de la Memoria de Patrones.
- **NOutputSignals:** Ancho de Bits de la Memoria de Patrones.
- **NDelays:** Tamaño de la Memoria de Retardos.
- **DelayWidth:** Ancho de Bits de la Memoria de Retardos.
- **NInputSignals:** Número de Entradas Externas.
- **MaxPrescale:** Valor máximo del Prescalado.
- **QueueSize:** Tamaño de la Cola de Eventos.

Puertos:

- **Clk (Entrada):** Señal de Reloj.
- **Reset (Entrada):** Señal de Reinicio Síncrono.
- **Enable (Entrada):** Señal de Activación.
- **WE_DelayTable (Entrada):** Señal de Escritura de la Memoria de Retardos.
- **WE_PatternTable (Entrada):** Señal de Escritura de la Memoria de Patrones.
- **Load_Prescale (Entrada):** Señal de Escritura del Registro del Prescalado.
- **Address_PatternTable (Entrada):** Dirección de Escritura de la Memoria de Patrones.
- **Address_DelayTable (Entrada):** Dirección de Escritura de la Memoria de Retardos.
- **Data (Entrada):** Datos de Escritura en las Memorias.
- **Input (Entrada):** Entradas Externas.
- **Enqueue (Entrada):** Señal de Puesta en Cola
- **Instruction (Entrada):** Instrucción actual a introducir en la Cola.
- **STOP (Salida):** Señal de Pausado de la Ejecución.
- **Output (Salida): Salida** de Patrones.
- **Overflow (Salida):** Señal de Error de Cola Rebosada.
- **Underflow (Salida):** Señal de Error de Cola Vacía.
- **Full_Queue (Salida):** Señal indicadora de Cola Llena.

Descripción Técnica

Su función es la de ejecutar las instrucciones que llegan desde el Generador de Secuencias. Se compone de la Cola de Eventos y el Generador de Eventos del diseño original. Su diagrama de bloques se observa en la Figura 7.

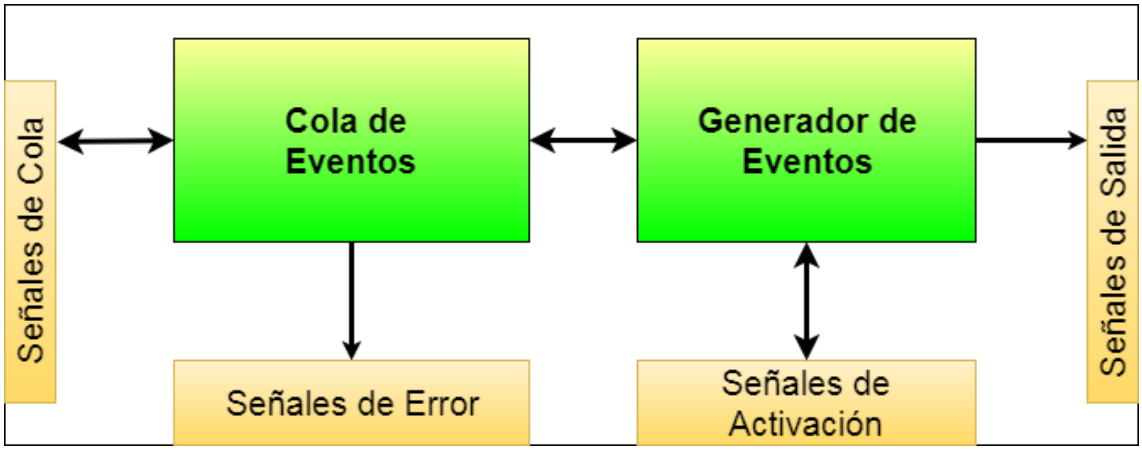

Figura 7. Diagrama de Bloques del Controlador de Eventos.

El Controlador de Eventos se compone de dos elementos principales:

- **La Cola de Eventos**, que recibe instrucciones nuevas desde las Señales de Cola, provenientes del Generador de secuencias y las almacena en una memoria FIFO.
- **El Generador de Eventos**, que lee las instrucciones de la Cola y las procesa una a una.

Cola de Eventos

Se trata de una memoria FIFO o cola, la cual almacena las Instrucciones de Generación de Eventos que el Generador de Secuencias le envía, y éstas son extraídas y procesadas posteriormente por el Generador de Eventos.

- **Introducir una nueva Instrucción:** Se cuenta con dos señales, la señal de entrada de datos que contiene la instrucción a introducir, y la señal de encolado, de 1 bit, que es la que introduce la instrucción al fondo de la Cola. Ambas son provenientes del Generador de Secuencias.

- **Extraer una Instrucción existente:** Se cuenta con dos señales, la señal de salida de datos que tiene el valor de la instrucción al principio de la cola, y la señal de desencolado, de 1 bit, que es la que elimina dicha instrucción. Ambas conectan con el Generador de Eventos.

La cola cuenta con una señal indicadora de cola llena, la cual se envía al Generador de Secuencias como señal bloqueante de la ejecución para evitar rebosar la cola, como se ha descrito en el apartado 4.1

La cola cuenta además con dos señales de error:

- **Overflow:** Desbordado de la Cola, es decir, se ha intentado guardar un nuevo dato cuando la Cola estaba llena.
- **Underflow:** Se ha intentado extraer un dato cuando la Cola estaba vacía.

Generador de Eventos

Nombre del Archivo: Event_Generator.vhd

Nombre de la Entidad: Event_Generator.

Generador de Eventos: Interfaz

Parámetros Genéricos:

- **InstructionWidth:** Ancho de Bits de las Instrucciones.
- **NPatterns:** Tamaño de la Memoria de Patrones.
- **NOutputSignals:** Ancho de Bits de la Memoria de Patrones.
- **NDelays:** Tamaño de la Memoria de Retardos.
- **DelayWidth:** Ancho de Bits de la Memoria de Retardos.
- **NInputSignals:** Número de Entradas Externas.
- **MaxPrescale:** Valor máximo del Prescalado.

Puertos:

- **Clk (Entrada):** Señal de Reloj.
- **Reset (Entrada):** Señal de Reinicio Síncrono.
- **Enable (Entrada):** Señal de Activación.
- **Instruction_In (Entrada):** Instrucción actual a ejecutar.
- **WE_DelayTable (Entrada):** Señal de Escritura de la Memoria de Retardos.
- **WE_PatternTable (Entrada):** Señal de Escritura de la Memoria de Patrones.
- **Load_Prescale (Entrada):** Señal de Escritura del Registro del Prescalado.
- **Address_PatternTable (Entrada):** Dirección de Escritura de la Memoria de Patrones.
- **Address_DelayTable (Entrada):** Dirección de Escritura de la Memoria de Retardos.
- **Prescale (Entrada):**
- **Pattern (Entrada):**
- **Delay (Entrada):**
- **Input (Entrada):**
- **Empty_Queue (Salida):**
- **Dequeue (Salida):**
- **STOP (Salida):**
- **Output (Salida):**

El Generador de Eventos se encarga de extraer Instrucciones de Generación de Eventos de la Cola de Eventos y ejecutarlas. Su diagrama de bloques se observa en la Figura 8.

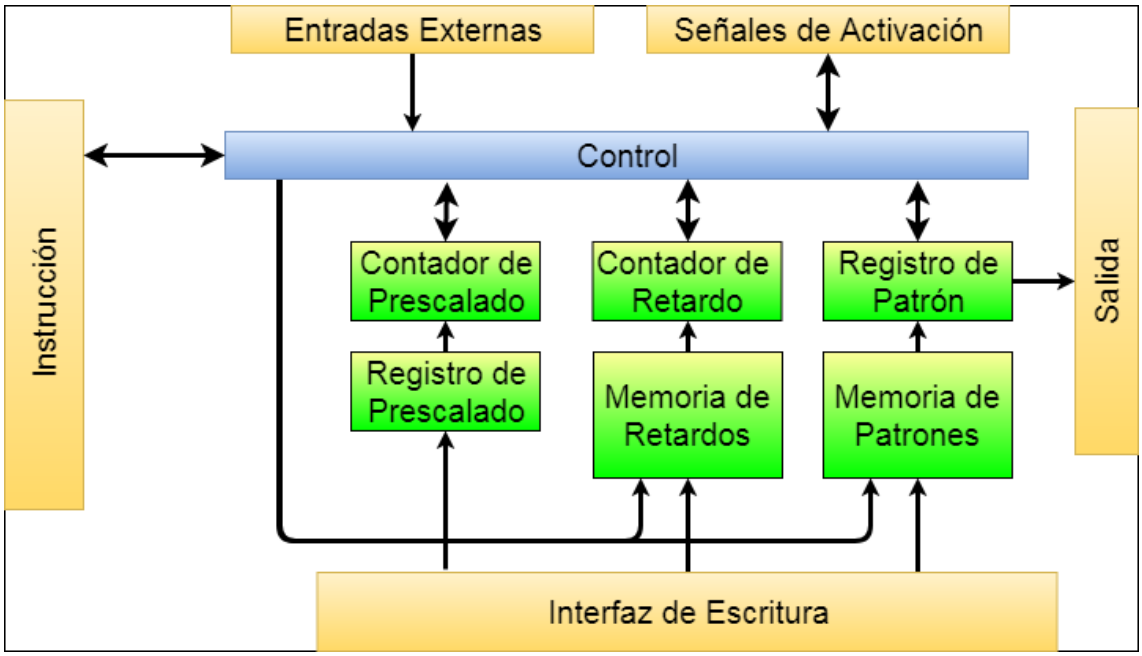

Figura 8: Diagrama de Bloques del Generador de Eventos.

- **Memorias Internas:** Se trata de un conjunto de dos memorias RAM, una que almacena patrones y otra que almacena retardos. Además se cuenta con un registro que almacena el valor del prescalado. El tamaño de estas memorias es el configurado en los parámetros genéricos del diseño.

  Cada uno de los tres elementos tiene su señal de escritura y su señal de dirección de memoria (salvo el registro del prescalado, pues sólo tiene una dirección). Los tres elementos comparten la misma señal de entrada de datos, cada uno recibiendo los bits correspondientes menos significativos (Ej.: El registro de prescalado recibe los 4 bits menos significativos de la entrada de datos). Estas señales provienen de la Interfaz de Escritura.

  Las Memorias de Retardos y Patrones también cuentan con una señal de dirección de lectura.

- **Registros y Contadores:** Se dispone de un registro para patrones (que almacenará el valor actual del patrón a escribir en la salida) y de dos contadores, uno para retardos y otro para prescalado. Estos dos contadores cuentan hacia abajo y están conectados en cascada.

Cada ciclo del reloj el contador de prescalado disminuye en uno, cuando intenta bajar por debajo de cero, se reinicia a su valor inicial y disminuye en uno el contador de retardos.
De esta manera, se alcanza el valor cero en ambos contadores tras un tiempo t = Retardo * Prescalado ciclos, donde Retardo y Prescalado son los valores iniciales de los contadores. Estos contadores disponen de una señal de activación para pausarlos.

- **Procesado de Instrucciones:** Cuando ambos contadores (Retardo y Prescalado) tienen valor cero, se procesa la instrucción del principio de la cola en el ciclo siguiente, de esta manera no se pierde un ciclo procesando la instrucción. La funcionalidad depende del tipo de instrucción:

    - **Instrucciones MOV:**
      Se lee el patrón y el retardo de sus respectivas memorias, cuyas direcciones de memoria son las indicadas por los argumentos de la instrucción. Sus valores son guardados en el Registro de Patrones y el Contador de Retardos, respectivamente, mientras que el Contador de Prescalado se reinicia al valor del prescalado almacenado en el Registro de Prescalado. La instrucción MOV se extrae de la cola y se descarta.

      La señal de salida es igual al patrón almacenado el Registro de Patrones, y cada ciclo se reducen los contadores de la manera especificada anteriormente. Tras el tiempo indicado se vuelve al valor cero en ambos contadores y se procesa la siguiente instrucción en la cola.

    - **Instrucciones SYNC:**
      Se dispone de un circuito lógico que compara la entrada y los argumentos de la instrucción para comprobar que son los adecuados, (La entrada externa indicada por la instrucción es igual al valor indicado por la misma).
      En el caso de que concuerden, se extrae la instrucción de la Cola, en el caso opuesto se mantiene en la Cola.
      En ambos casos se mantendrá el valor de ambos contadores a cero, para que el ciclo siguiente se vuelva a procesar la instrucción del tope de la cola.

      De esta manera, el retardo mínimo causado por la instrucción SYNC (en el caso de que la entrada externa tuviese el valor adecuado desde el principio) es de un ciclo de reloj. Así mismo el retardo entre un cambio en la entrada al valor correcto y la detección de este es siempre de un ciclo de reloj, permitiendo que el comportamiento del procesador sea predecible.

Las Memorias de Retardos y Patrones, así como los Contadores de Retardo y Prescalado, se modelarán mediante los componentes genéricos “RAM Genérica” y “Contador Genérico” que se describen más adelante en el apartado 4.3.6.

Observaciones

Los Parámetros Genéricos se heredan de la entidad superior, Generador de Eventos, por lo que si se configuran mal éstos de acuerdo con las observaciones del apartado 4.4, el Generador de Eventos no funcionará correctamente.

Las instrucciones MOV hacen referencia a una dirección de la Memoria de Patrones y a otra de la Memoria de Retardos. Debido a esto, se puede dar el caso de que la instrucción haga referencia a una dirección de memoria en la que no se haya escrito un dato previamente, lo que implicaría la salida de un patrón indeterminado durante un tiempo indeterminado. Para evitar esto se debe tener precaución en la programación del ASIP.

Observaciones

Los Parámetros Genéricos se heredan de la entidad superior, ASIP, por lo que si se configuran mal éstos de acuerdo con las observaciones del apartado 4.4, el Generador de Eventos no funcionará correctamente.

Se debe tener en cuenta una serie de restricciones en el programa para evitar los errores de ejecución.

- Debe de haber coherencia entre los argumentos de las instrucciones MOV y los datos cargados con el programa, es decir, no referenciar mediante la instrucción MOV una dirección de Retardo o Patrón que no se haya escrito previamente.
- A la hora de escribir los programas, se debe tener en cuenta que la base de tiempo del Generador de Eventos es distinta debido a la existencia del prescalado. Un Retardo de valor 1 equivale a un número de ciclos de reloj iguales al valor del Prescalado.
- Se debe considerar la disparidad entre la velocidad a la que se reciben nuevas instrucciones en la Cola (que depende del tipo de instrucción que esté procesando el Generador de Secuencias), y la velocidad a la que se ejecutan éstas (que depende de los Retardos y el Prescalado en el caso de las instrucciones MOV y del usuario en el caso de las instrucciones SYNC). Debido a esto, se debe evitar que las instrucciones se retiren de la Cola más rápido de lo que entran, pues podría provocar un Error de Cola Vacía si se consigue vaciar la Cola por completo. Una solución rápida a este problema es aumentar el tamaño de la Cola.

4.3.6 Componentes Genéricos

Se han modelado dos componentes de uso genérico para su uso en varias partes del diseño. Estos componentes son los siguientes:

RAM Genérica

Nombre del Archivo: GenericRAM.vhd

Nombre de la Entidad: GenericRAM.

Parámetros Genéricos:

- **WordSize:** Ancho de bits de cada dirección de memoria.
- **MemorySize:** Número de direcciones de memoria.

Puertos:

- **Clk (Entrada):** Señal de Reloj.
- **WE (Entrada):** Señal de Escritura.
- **Address_Write (Entrada):** Dirección de Escritura de la Memoria.
- **Address_Read (Entrada):** Dirección de Lectura de la Memoria.
- **Data_in (Entrada):** Datos escritos a la Memoria.
- **Data_out (Salida):** Datos leídos de la Memoria.

Funcionamiento:

Consiste de una memoria RAM cuyo tamaño es especificado por los parámetros genéricos WordSize (ancho de bits de cada registro) y MemorySize (número total de direcciones de la memoria).

Cuenta con dos puertos, uno de escritura y otro de lectura. El puerto de escritura es usado durante la programación inicial del ASIP y el de lectura durante la ejecución del programa.

Contador Genérico

Nombre del Archivo: GenericCounter.vhd

Nombre de la Entidad: GenericCounter.

Parámetros Genéricos:

- **MaxCount:** Valor Máximo alcanzable por el contador.

Puertos:

- **Clk (Entrada):** Señal de Reloj.
- **Reset (Entrada):** Señal de Reinicio Asíncrono.
- **Enable (Entrada):** Señal de Activación
- **Count_down (Entrada):** Señal de Dirección de Cuenta
- **Load (Entrada):** Señal de Escritura.
- **Data_in (Entrada):** Datos de Escritura.
- **Count (Entrada):** Valor actual del Contador.

Funcionamiento:

Consiste de un contador simple que incrementará su valor de uno en uno hasta el valor MaxCount.

El Contador toma el valor 0 al activar Reset, y sólo cambiará de valor de manera síncrona cuando Enable esté activa.

Se puede cambiar la dirección de cuenta del Contador mediante la señal Count_Down. Esta señal está activa a nivel alto, por lo que si está inactiva la cuenta es ascendente, y si está activa la cuenta es descendente.

El Contador permite la carga de un valor directamente mediante la señal Load. Este proceso tiene mayor prioridad que el incremento normal.

4.3.7 Control de Activación

El Control de Activación es un conjunto de lógica que lidia con las señales de activación y parada para pausar o resumir la ejecución del programa.

El ASIP cuenta con tres mecanismos de parada general y un mecanismo para pausar el Generador de Secuencias:

- **Inicio de la Ejecución:**
  La ejecución de ambas partes del ASIP se mantiene en pausa mediante señales de bloqueo hasta que el usuario la inicie mediante un interruptor externo, START. Esto se debe a que se deben escribir los datos de programa antes de iniciar la ejecución.

  Para realizar este bloqueo se dispone de dos registros de un bit, o flags, uno para el Generador y otro para el Controlador. Si el flag tiene el valor 0, el bloque correspondiente pausa la ejecución.

  Una vez se active la entrada START, el flag del Generador tomará el valor 1 y, si no hay otro bloqueo aplicado, empezará a funcionar. Cuando la entrada START este activa y se introduzca la primera Instrucción de Generación de Eventos en la Cola de Eventos, el flag del Controlador tomará el valor 1. Con esto se evita que el Controlador encuentre una cola vacía al principio de la ejecución.

  Aunque se desactive la entrada START, una vez estos flags estén activos permanecerán activos hasta el reinicio del ASIP para evitar la parada manual por parte del usuario.

- **Bloqueo por Instrucción SYNC:**
  El Controlador de Eventos emite una señal STOP, que estará activa cuando la señal SYNC lo determine. Mientras esta señal esté activa se pausará la ejecución del Generador y el Controlador, salvo la comprobación de la señal SYNC cada ciclo.

- **Bloqueo por Error:**
  En el caso de detectarse cualquier Señal de Error activa se parará la ejecución permanentemente.

- **Bloqueo por Cola Llena:**
  Este mecanismo de bloqueo solo pausa la ejecución del Generador de Secuencias.
  El Controlador de Eventos transmite una señal de Cola Llena al Control de Activación. En el caso de que esta señal este activa, el Generador pausara la ejecución. De esta manera solo se procesan instrucciones cuando hay espacio en la cola y se evita un desborde de ésta.

4.3.8 Señales de Error

El ASIP cuenta con una función de detección de errores que detecta hasta 6 errores distintos. Cada señal de error generada por los componentes donde se detecte éste activará un flag indicador de estado de error. El ASIP cuenta con siete salidas, una por error, conectadas a estos indicadores.

Una vez detectado un error se activará su indicador correspondiente y se parará la ejecución permanentemente hasta el reinicio del ASIP. Cada señal es independiente por lo que se puede detectar más de un error a la vez, sin embargo, es muy poco probable que se dé el caso debido a que la ejecución se para tras detectar un error, por lo que deberían ocurrir simultáneamente.

Las señales de error están posicionadas de manera ascendente de 0 (bit menos significativo) a 5 (bit más significativo), como se describe en la Tabla 13.

Posición	Error
0	**Dirección no Válida**
1	**Datos no Válidos**
2	**Instrucción no Válida**
3	**Pila Rebosada**
4	**Pila Vacía**
5	**Cola Vacía**

Tabla 13: Señales de Error.

A continuación se describen los errores en detalle:

Errores producidos al escribir los Datos del Programa:

- **Dirección no Válida:**
  Este error se produce cuando una dirección está fuera del rango permitido.

  Este error depende de las secciones de memoria dedicadas a cada memoria interna del ASIP y por tanto de los Parámetros Genéricos.

- **Datos no Válidos:**
  Este error se produce cuando un valor está fuera del rango permitido para sección de memoria indicada por la dirección.

  Este error depende del ancho de bits de las memorias internas del ASIP así como del valor máximo del prescalado y por tanto de los Parámetros Genéricos.

Errores producidos durante la Ejecución

- **Instrucción no Válida:**
  Este error se produce al intentar decodificar una instrucción cuyo Código de Operación no se reconoce.

Este error depende del Conjunto de Instrucciones del ASIP por lo que no depende de los Parámetros Genéricos.

- **Pila Rebosada:**
  Este error se produce al superar el número máximo de Niveles de Llamada en la Pila, es decir, al tener demasiados bucles, saltos, o llamadas a subrutinas anidados.

  Este error depende del tamaño de la Pila del ASIP, y por tanto, de los Parámetros Genéricos, así como del Programa Cargado.

- **Pila Vacía:**
  Este error se produce al intentar eliminar un Nivel de Llamada de la Pila estando está ésta vacía, es decir, ejecutando una instrucción de retorno sin haber creado un bucle o subrutina primero.

  Este error depende sólo del Programa Cargado, por lo que no depende de los Parámetros Genéricos.

- **Cola Vacía:**
  Este error se produce al intentar procesar la siguiente instrucción de la Cola cuando ésta está vacía. Este error se debe a un Programa Cargado poco apropiado, teniendo éste demasiadas Instrucciones de Control seguidas, por lo que se procesan las Instrucciones de Generación de Eventos más rápido de lo que se introducen en la Cola.

  Este error depende sólo del Programa Cargado, por lo que no depende de los Parámetros Genéricos.

4.4 Observaciones Generales

4.4.1 Arquitectura

Los valores de los parámetros genéricos deben ser coordinados de manera correcta para que el sistema funcione. Los factores que se tienen que tomar en cuenta son los siguientes:

- **Ancho de Instrucción:**
  Los Códigos de Operación de las instrucciones son de 2, 3 y 4 bits, por lo que el ancho de sus parámetros será el resto de bits de la instrucción. De esta manera, el valor de sus parámetros está determinado por el ancho de instrucción. Por ejemplo, con el ancho por defecto de 16, las instrucciones CALL, JUMP y DECRET solo pueden acceder a una instrucción de 13 bits, y por tanto a una dirección de memoria de hasta 8192.

- **Número de Salidas:**
  El número de salidas determina el ancho de datos de escritura al ASIP y por tanto debe ser siempre mayor que el Ancho de Instrucción y que el Ancho de Retardos.

- **Número de Patrones y Número de Retardos:**
  Estos determinan el tamaño de sus respectivas memorias, por lo que si se cambian se debe ajustar el ancho de instrucción, pues la instrucción MOV tiene 14 bits por defecto para las direcciones, 8 para patrones y 6 para retardos. Estos valores coinciden con sus tamaños por defecto de 256(2^8) y 64(2^6) respectivamente. De esta manera, el nuevo ancho de instrucción debe ser:

  $$Ancho\ Instrucción\ \geq 2 + \log_2 NPatrones + \log_2 NRetardos$$

- **Ancho de Retardos:**
  Debe ser menor que el Número de Salidas para poder escribirse correctamente en la interfaz de escritura.

- **Número de Instrucciones:**
  Se usa como la mitad del mapa de memoria, por lo que su tamaño debe ser mayor al tamaño de resto de memorias (Patrones, Retardos y Prescalado). Por otra parte, el ancho de instrucciones se debe reajustar para poder acceder a las nuevas direcciones de instrucción, como se ha descrito anteriormente.

  $$NInstrucciones \geq 2 * (NRetardos + NPatrones + 1)$$
  $$Ancho\ Instrucción\ \geq= 3 + \log_2 NInstrucciones$$

- **Máximas Iteraciones:**
  Limita el valor del argumento de las instrucciones correspondientes, por las que el Ancho de instrucción debe ser ajustado correctamente.

  $$Ancho\ Instrucción\ \geq= 4 + \log_2 MaxIteraciones$$

- **Valor Máximo del Prescalado:**
  Este valor afectará al retardo mínimo y máximo como se explicará más adelante.

- **Tamaño de Pila y Tamaño de Cola:**
  El valor de estos parámetros limitarán los programas validos como se explicará más adelante.

- **Número de Entradas:**
  El valor no tiene efecto observable en el programa por debajo de 2048. Al ser estas entradas externas ese valor nunca se alcanzará en la práctica.

- **Errores por Cola Desbordada:**
  Se han omitido indicador de error para este caso debido a que el diseño del hardware hace imposible que se dé, dado que el Generador de Secuencia pausa su ejecución si la Cola está llena.

4.4.2 Temporización y Sincronizado

- **Tiempos Límite de Retardo:**

  Se deben tener en cuenta las limitaciones en los tiempos reales de salida de cada patrón en base a los valores de los retardos y del prescalado.

  El tiempo de permanencia de un patrón en la salida del ASIP es igual al valor de su retardo multiplicado por el prescalado. Por lo tanto los valores límite de este tiempo son los siguientes:

  $t_{min} = Retardo_{min} * Prescalado_{min} = 1 * 1 = 1\ ciclo\ de\ reloj$
  $t_{max} = Retardo_{max} * Prescalado_{max} = \log_2 AnchoRetardo * MaxPrescalado$

  Para valores por defecto de los parámetros genéricos: $t_{max} = 10.485.760\ ciclos\ de\ reloj$

- **Tiempo de Sincronizado:**

  El tiempo mínimo que la instrucción SYNC puede parar la ejecución es de un ciclo de reloj, siendo el tiempo máximo infinito.

- **Vaciado de la Cola:**

  Es posible que durante la ejecución se dé el caso de que la Cola de Eventos se vacíe, produciendo un error. Esto se debe a que las instrucciones se procesan más rápido de lo que se leen de la memoria, y en ese caso se debe revisar el programa.

4.4.3 Programa

Se deben de tener en cuenta varios aspectos de este diseño a la hora de escribir un programa.

- **Parámetros Genéricos:**
  Se debe asegurar que los valores de los parámetros genéricos del código fuente del Ensamblador coinciden con los valores del ASIP, y se debe asegurar que el programa respete esos valores (Ej. El ancho de bits de cada patrón debe ser menor o igual al valor del parámetro genérico NSalidas).

- **Niveles de llamada:**
  El número de llamadas a subrutinas anidadas está determinado por el parámetro genérico TamañoPila, siendo el límite uno más de este valor, 9 por defecto. En el caso de tener más llamadas anidadas de las permitidas se producirá un error, por lo que se debe alterar el código o el parámetro genérico correspondiente.

- **Vaciado de Cola:**
  Como se ha expuesto en el apartado anterior, es posible que se dé el caso de que la ejecución de las instrucciones sea más rápida que su búsqueda, produciendo un error, por lo que se debe aumentar el tamaño de la cola o revisar el programa.

  Cada instrucción tarda un ciclo en buscarse y ser ejecutada por el Generador de Secuencias, pasando a la Cola de Eventos sólo si es una instrucción MOV o SYNC. Las instrucciones MOV tardarán en ejecutarse un número de ciclos igual al valor de su retardo multiplicado por el valor del prescalado, mientras que las instrucciones SYNC tienen un tiempo de ejecución indeterminado. Teniendo esto en cuenta se debe revisar el programa para asegurar que no hay demasiadas instrucciones de salto seguidas que pudiesen provocar que se vaciase la cola.

- **Delay Slot:**
  Debido a la ejecución en etapas de las instrucciones debemos considerar el uso del Delay Slot para cada instrucción como se ha especificado anteriormente. Se debe programar con esto en cuenta y añadir la instrucción LDC o NOP después de cada instrucción CALL. No hacerlo puede provocar un orden de ejecución no deseado.

5. Entorno de Desarrollo para el ASIP

Para facilitar la programación del ASIP se ha creado un Entorno de Desarrollo consistente de un ensamblador, es decir, una aplicación que compila el lenguaje ensamblador del ASIP en lenguaje de máquina (código binario) directamente ejecutable por el ASIP.

A continuación se describirá las especificaciones del Lenguaje Ensamblador del ASIP.

5.1 Lenguaje Ensamblador

El Lenguaje Ensamblador que se ha usado en el ASIP está dividido en cuatro secciones claramente diferenciables, identificados por una cabecera. Estas cabeceras son de uso obligatorio. Todos los valores numéricos del código serán escritos en decimal. Las distintas cabeceras se observan en la Tabla 14.

Sección	Cabecera	Descripción
Prescalado	#Prescale	Valor del Prescalado
Patrones	#Pattern Table	Listado de los Patrones a usar durante la ejecución
Retardos	#Delay Table	Listado de los Retardos a usar durante la ejecución
Instrucciones	#Code	Programa (Lista de Instrucciones)

Tabla 14: Cabeceras del Código Ensamblador.

A continuación se describen en detalle las secciones:

- **Prescalado:**
  Consiste de una sola línea, el valor en decimal del prescalado.

- **Patrones y Retardos:**
  Consisten de un listado de los patrones o retardos que serán utilizados por el programa. Los listados deben ser numerados, es decir, cada línea consistirá del índice del patrón o retardo a usar seguido del valor, separados por un espacio. El valor de los retardos deberá ser escrito en decimal, mientras que el valor de los patrones será escrito en binario.
  Para los patrones, en el caso de tener estos menos bits que el total posible determinado por el parámetro NSalidas (por defecto, 32), los valores se justificarán a la derecha (bits menos significativos), rellenando con ceros los bits no usados. En cualquier caso el ancho de bits de todos los patrones deberá ser el mismo.

- **Instrucciones:**
  Se trata de un listado de las instrucciones del programa a ejecutar, escritas mediante su código de instrucción seguido de sus argumentos (descritos en el conjunto de instrucciones) separados por comas (ej. “MOV 3, 1”).
  Se permite añadir etiquetas para identificar secciones de las instrucciones como subrutinas o puntos de entrada de saltos, las etiquetas consistirán de un nombre

seguido inmediatamente de dos puntos (ej. “Inicio:”). Las instrucciones cuyo argumento consista en una dirección de memoria (CALL, DECRET y JUMP) deberán usar como argumento una etiqueta en lugar de un valor (ej. “JUMP Inicio”).

Se permite el espaciado entre líneas, el sangrado y el espacio en blanco entre palabras o al final de una línea, salvo las comas de separación de argumentos que deben ser situadas inmediatamente después del argumento, al igual que los dos puntos de las etiquetas.

Se pueden añadir comentarios que serán ignorados por la aplicación mediante el uso del punto y coma, de tal manera que cualquier código desde un punto y coma hasta la siguiente línea será ignorado (ej. “; comentario”).

5.2 Aplicación Ensambladora

5.2.1 Descripción

La Aplicación Ensambladora, o Ensamblador, consiste en una sencilla aplicación de consola que leerá un archivo de texto “Program.txt”, situado en la misma carpeta que la propia aplicación, que contenga el programa en código ensamblador a compilar. La aplicación se ha desarrollado en el lenguaje de programación C# debido a su comodidad.

El Ensamblador tendrá dos modos de operación, modo “Standalone” y modo “Periférico AXI”. Al comienzo de la ejecución el usuario deberá elegir una de las dos opciones.

Modo Standalone:

Este modo produce un fichero de texto conteniendo el Código de Máquina a usar directamente por el ASIP.
Tras procesar el fichero Program.txt, se generará otro fichero de texto, “MachineCode.txt”, en la misma carpeta, que contendrá las líneas de código de máquina para el ASIP. Si el archivo MachineCode.txt ya existe se sobrescribirá.
Cada línea consistirá de la dirección de escritura en binario inmediatamente seguida por el dato correspondiente a esa dirección, también en binario. El ancho de cada una de estas dos partes, y por tanto de la línea completa, dependerá de los valores de los parámetros genéricos configurables. Para los valores por defecto, cada línea de programa compilado tendrá la estructura descrita en la Figura 9, donde se observa el número de dígitos binarios de cada línea y el dato que representa cada uno. Se adjunta un ejemplo del Código de Máquina resultante en el Anexo 2.

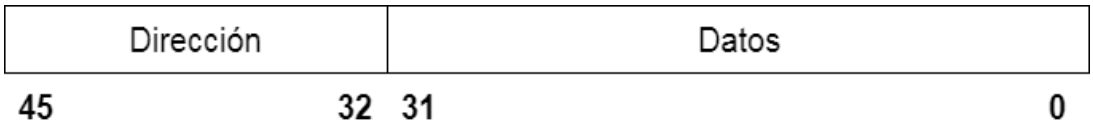

Figura 9: Codificación del Código de Maquina

Modo Periférico AXI:

Como se explicará más adelante en el apartado 7.2, el ASIP está pensado para trabajar en conjunto con un microprocesador a través del cual se escribirán los datos de programa. Debido a esto el Ensamblador cuenta con la opción de generar directamente el código fuente en el lenguaje de programación C a ser ejecutado por el microprocesador. El Ensamblador leerá el archivo Program.txt y generará un fichero Main.c que podrá ser directamente ejecutable por el Entorno de Desarrollo Software como se explicará más adelante en el apartado 7.4. Se puede observar un ejemplo del fichero Main.c resultante en el Anexo 2.

Observaciones:

Debido al carácter genérico del ASIP, el Ensamblador también dependerá de los mismos parámetros genéricos. El Ensamblador toma como valores los valores por defecto, si se requiere cambiarlos será necesario cambiarlos en el código fuente de la aplicación y recompilarla.

La aplicación cuenta con una función de detección de errores de sintaxis para asegurar que el código de máquina tiene el formato correcto. Además de esta detección, la aplicación cuenta con una función de detección de errores de valores, es decir, detecta que valores de los argumentos, a pesar de ser perfectamente ejecutables, producirían un error de ejecución en el ASIP más adelante. Esta detección de errores de valores, a pesar de ser redundante con la detección de errores del ASIP, permite encontrar el error antes de cargar el programa en el dispositivo.

La lista de errores que detecta el Ensamblador y sus descripciones se observa en la Tabla 15.

Error	Tipo	Descripción
Archivo no Encontrado	Error de Fichero	No se ha encontrado el archivo “Program.txt” en la carpeta.
Cabecera no Válida	Error de Sintaxis	Se ha escrito una cabecera no permitida.
Número de Argumentos no Válido	Error de Sintaxis	El número de argumentos de una instrucción no es el correcto.
Dirección no Válida	Error de Sintaxis	Formato no válido al introducir una dirección.
Dato no Válido	Error de Sintaxis	Formato no apropiado para ese dato.
Más de un Prescalado	Error de Sintaxis	Se ha escrito más de un valor de Prescalado
Tamaño de Salida no Constante	Error de Sintaxis	Los patrones del Listado de Patrones no tienen un ancho de bits constante.
Nombre de Etiqueta no Válido	Error de Sintaxis	Formato no válido al introducir una dirección.
Etiqueta Repetida	Error de Sintaxis	Se ha definido una etiqueta dos veces.
Instrucción no Definida	Error de Sintaxis	Se ha escrito una instrucción no incluida en el conjunto de instrucciones del ASIP
Máximo de Instrucciones Superado	Error de Valores	Se ha superado el máximo de instrucciones soportadas por el ASIP.
Dirección Fuera de Rango	Error de Valores	El valor de la dirección está fuera del rango valido para esa sección de memoria.

Dato Fuera de Rango	Error de Valores	El valor del dato está fuera del rango valido para ese tipo de dato.
Nivel de Llamada Máximo Superado	Error de Valores	Se ha superado el número máximo de niveles de llamada (bucles anidados).
Excepción no Controlada	Error del Compilador	Se ha producido un error no especificado durante la compilación.

Tabla 15: Casos de Error de la Aplicación Ensambladora

5.2.2 Pruebas

Para asegurar un correcto funcionamiento del Ensamblador se han realizado una serie de pruebas. Se considerarán dos tipos de pruebas:

- **Prueba de Funcionamiento Correcto:** Para realizar esta prueba se ha intentado compilar el Programa de Prueba, adjunto en el Anexo 1. Este programa cuenta con todas las instrucciones distintas, por lo que prueba la funcionalidad completa del Ensamblador salvo los casos de Detección de Errores. Tras realizar la prueba el Ensamblador ha compilado el programa correctamente sin ningún tipo de error.

- **Pruebas de Detección de Errores:** Para realizar estas pruebas se ha creado sistemáticamente una serie de Programas que contienen cada uno de los Casos de Error especificados en el apartado anterior, en la Tabla 14, y se ha intentado compilar cada uno de ellos. Tras realizar las pruebas el Ensamblador ha presentado el mensaje de error adecuado en cada uno de estos casos.

6. Simulación y Pruebas

Para comprobar el funcionamiento de ASIP se han realizado una serie de simulaciones que prueban las distintas funcionalidades de éste. Las simulaciones se han realizado con el software Xilinx Vivado 2016.1, una herramienta de simulación y síntesis lógica.

Las simulaciones se han realizado mediante el método de "testbench" o banco de pruebas, el cual consiste en simular el comportamiento de un diseño HDL, al que llamaremos banco de pruebas, que encapsula el diseño a probar. El circuito banco de pruebas contiene una serie de señales que se conectarán a los puertos del ASIP. A estas señales se les asignará formas de onda definidas por el usuario, de tal manera que se pueda simular cualquier conjunto de señales de entrada para comprobar todos los casos de uso.

Estas formas de onda requieren especificación de tiempos concretos por lo que no son sintetizables, es decir, solo se pueden usar en simulación y no formarán parte del diseño final del ASIP, por lo que las pruebas físicas deberán usar un método distinto.

Con la herramienta Vivado se puede observar tanto el valor de las entradas y salidas del ASIP (las señales del banco de pruebas) como los valores de las señales internas del ASIP y sus componentes internos, por lo que se puede comprobar todas las señales individualmente para asegurar el correcto funcionamiento del ASIP.

El conjunto de pruebas que se han realizado se dividen en dos grupos:

- **Pruebas de Funcionalidad:**
  Pruebas que comprueban el comportamiento correcto del ASIP como conjunto y de sus componentes individuales.

- **Pruebas de Detección de Errores:**
  Pruebas que comprueban los casos límite del diseño y aseguran que se producen señales de Error correctamente.

6.1 Pruebas de Funcionalidad

Estas pruebas consisten en comprobar el funcionamiento de cada componente individual así como el del diseño completo del ASIP en condiciones de operación típicas, es decir, la ejecución de un programa que no viole ninguna de las restricciones de programación descritas en el apartado 4.4.3. El programa de prueba en concreto es el Programa de Prueba 1, adjunto en el Anexo 1.

Todas las Pruebas de Funcionalidad comparten el mismo banco de pruebas, que consiste en cargar el Programa de Prueba 1.

6.1.1 Banco de Pruebas

El Banco de Pruebas usado para todas las Pruebas de Funcionalidad consiste de la carga del Programa de Prueba en el ASIP, compilado mediante el Ensamblador, seguido del simulado de las Entradas Externas por parte del usuario. El programa se introduce en el banco de pruebas como una memoria ROM conteniendo las líneas del programa.

Se simula una señal de Reloj con un periodo de 10ns para simular una frecuencia de reloj de 100MHz.

El proceso que se sigue es el siguiente:

1. **Configuración Inicial:** Se activa la Señal de Reinicio y se desactiva el resto (Entradas Externas, Señal de Escritura y Señal de START). Con esto se logra reiniciar el ASIP a su configuración base.

2. **Escritura de Datos:** Se desactiva la Señal de Reinicio y se activa la Señal de Escritura. Las señales de Datos y Dirección toman los valores de cada línea de programa, a una frecuencia de una línea de programa por ciclo de reloj. Con este proceso se consigue escribir el Programa de Prueba en las memorias internas del ASIP.

3. **Simulación de Señales Externas:** Una vez escrita la última línea del programa se desactiva la Señal de Escritura y se activa la Señal de START, comenzando la ejecución. Más tarde se activa la señal de Entrada Externa correspondiente a la instrucción SYNC del Programa de Prueba. Con esto se simula la interacción del usuario.

El Programa de Pruebas consiste en la generación de una imagen mediante una conexión VGA, la imagen en concreto un patrón de tablero de ajedrez. El programa comenzará con una instrucción SYNC (se pausará hasta que el usuario active la Entrada Externa correspondiente) y procederá a repetir los patrones de la imagen indefinidamente.

La comprobación consiste en observar los patrones de salida de simulación, y el tiempo entre ellos, y comprobar si coinciden con los especificados en el Programa de Prueba.

6.1.2 Casos de Prueba

Se consideran los siguientes tipos de pruebas, descritos en la Tabla 16.

Número	Nombre	Descripción
1	Programación Inicial	Comprobación del correcto funcionamiento de la Escritura de Datos de Programa inicial.
2	Control de Activación	Comprobación del correcto funcionamiento del Control de Activación.
3	Generador de Secuencias	Comprobación del correcto funcionamiento del Generador de Secuencias.
4	Cola de Eventos	Comprobación del correcto funcionamiento de la Cola de Eventos.
5	Controlador de Eventos	Comprobación del correcto funcionamiento del Controlador de Eventos.
6	Instrucción MOV	Comprobación del correcto procesado de la instrucción MOV.
7	Instrucción SYNC	Comprobación del correcto procesado de la instrucción SYNC.
8	Sistema Completo	Comprobación del correcto funcionamiento del ASIP como conjunto y su reacción a las Entradas Externas y de START.

Tabla 16: Tipos de Pruebas de Funcionamiento.

En base a estos tipos se consideran los siguientes casos de prueba:

Caso 1.1	
Nombre	**Programación de Datos Correcta**
Tipo	Programación Inicial
Resultado Esperado	Los datos de programa se guardan con su valor correcto en las direcciones indicadas por el programa.

Caso 1.2	
Nombre	**Programación de Datos Correcta**
Tipo	Programación Inicial
Resultado Esperado	El mapeo de memoria se realiza correctamente y cada bloque de direcciones solo guarda el dato en la memoria correcta.

Caso 2.1	
Nombre	**Control de Activación Correcto**
Tipo	Control de Activación
Resultado Esperado	Las Señales de Activación del Generador de Secuencias y el Controlador de Eventos se calculan correctamente como se describe en el apartado 4.3.7.

Caso 2.2	
Nombre	**Pausado Correcto**
Tipo	Control de Activación
Resultado Esperado	La ejecución del Generador y el Controlador se pausa y reanuda correctamente en función de las Señales de Activación

Caso 3.1	
Nombre	**Lectura de Instrucción Correcta**
Tipo	Generador de Secuencias
Resultado Esperado	La instrucción de la Memoria de Instrucciones (IM) determinada por el Contador de Programa (PC) se lee correctamente y se guarda en el Registro de Instrucciones (IR).

Caso 3.2	
Nombre	**Administración de Pila Correcta**
Tipo	Generador de Secuencias
Resultado Esperado	La Pila añade, edita y retira registros cuando las correspondientes señales se activan.

Caso 3.3	
Nombre	**Decodificación de Instrucción Correcta**
Tipo	Generador de Secuencias
Resultado Esperado	El Decodificador de Instrucciones genera las señales de control correctamente de acuerdo con el apartado 4.3.4

Caso 4.1	
Nombre	**Envío a Cola Correcto**
Tipo	Cola de Eventos
Resultado Esperado	La señal siendo procesada por el Generador de Secuencias es recibida por la Cola y se introduce sí y sólo sí ésta es una Instrucción de Generación de Eventos.

Caso 4.2	
Nombre	**Recepción en Cola Correcta**
Tipo	Cola de Eventos
Resultado Esperado	La señal siendo procesada por el Generador de Secuencias es recibida por la Cola y se introduce sí y sólo sí ésta es una Instrucción de Generación de Eventos.

Caso 5.1	
Nombre	**Lectura de Cola Correcta**
Tipo	Controlador de Eventos
Resultado Esperado	Las Instrucciones sólo se leen y procesan cuando ambos contadores tienen valor cero.

Caso 5.2	
Nombre	**Eliminado de Instrucción Correcto**
Tipo	Controlador de Eventos
Resultado Esperado	Cuando una Instrucción ha terminado de ejecutarse, se remueve de la cola.

Caso 6.1	
Nombre	**Lectura de Patrones y Retardos Correcta**
Tipo	Instrucción MOV
Resultado Esperado	Se lee correctamente el patrón y el retardo indicados de sus respectivas memorias y estos se guardan en el Registro de Patrón y el Contador de Retardos.

Caso 6.2	
Nombre	**Cuenta de Ciclos Correcta**
Tipo	Instrucción MOV
Resultado Esperado	El Contador de Retardos disminuye en uno cada número de ciclos igual al valor del Registro de Prescalado.

Caso 6.3	
Nombre	**Salida Correcta**
Tipo	Instrucción MOV
Resultado Esperado	La Salida de Datos es igual al valor del Registro de Patrón.

Caso 7.1	
Nombre	**Comprobación de Señal de Entrada Correcta**
Tipo	Instrucción SYNC
Resultado Esperado	Se comprueba correctamente si la Señal de Entrada Externa determinada por la instrucción coincide con el valor especificado.

Caso 7.2	
Nombre	**Generación de Señal de STOP Correcta**
Tipo	Instrucción SYNC
Resultado Esperado	Se genera la Señal de STOP correctamente y se mantienen en valor cero ambos contadores mientras esta comprobación sea negativa.

Caso 7.3	
Nombre	**Descarte de Instrucción SYNC Correcto**
Tipo	Instrucción SYNC
Resultado Esperado	Se descarta la instrucción cuando esta comprobación es positiva.

Caso 8.1	
Nombre	**Generación de Patrones Correcta**
Tipo	Sistema Completo
Resultado Esperado	Se generan las señales de salida de acuerdo al programa de prueba.

Caso 8.2	
Nombre	**Comienzo de Ejecución Correcto**
Tipo	Sistema Completo
Resultado Esperado	La ejecución comienza de acuerdo con la Señal de START.

Caso 8.3	
Nombre	**Parada y Reanudación de Ejecución Correctas**
Tipo	Sistema Completo
Resultado Esperado	La parada y reanudación de la ejecución mediante la instrucción SYNC se realiza correctamente de acuerdo con las Señales de Entrada externa.

Caso 8.4	
Nombre	**Detección de Errores Correcta**
Tipo	Sistema Completo
Resultado Esperado	No se generan señales de error.

Se han realizado las Pruebas de acuerdo con las especificaciones anteriores y se ha verificado el comportamiento adecuado del ASIP en cada uno de los casos.

6.2 Pruebas de Detección de Errores

Estas pruebas consisten en comprobar el funcionamiento de cada componente individual así como el del diseño completo del ASIP en condiciones de operación límite, es decir, la ejecución de un programa viole las restricciones de programación descritas en el apartado 4.4.3.

Todas las Pruebas usan el mismo Banco de Pruebas, descrito en el apartado anterior. Se carga un programa de prueba distinto para cada caso que viole la restricción de programación que cause ese error.

Se consideran los tipos de pruebas descritos en la tabla 17.

Número	Nombre	Descripción
1	Programación Inicial	Comprobación del correcto funcionamiento de la Escritura de Datos de Programa inicial.
2	Generador de Secuencias	Comprobación del correcto funcionamiento del Generador de Secuencias.
3	Controlador de Eventos	Comprobación del correcto funcionamiento del Controlador de Eventos.

Tabla 17: Tipos de Pruebas de Detección de Errores

Se consideran los siguientes casos de prueba:

Caso 1.1	
Nombre	**Escritura de Dirección Incorrecta**
Tipo	Programación Inicial
Condición de Prueba	Se debe cargar un programa que introduzca una dirección con un formato incorrecto (ej. Un valor no numérico).
Resultado Esperado	Se genera una Señal de Error de Dirección no Válida.

Caso 1.2	
Nombre	**Escritura de Datos Incorrecta**
Tipo	Programación Inicial
Condición de Prueba	Se debe cargar un programa que introduzca un dato con un formato incorrecto (ej. Un valor no binario).
Resultado Esperado	Se genera una Señal de Error de Dato no Válido.

Caso 2.1	
Nombre	**Decodificación de Instrucción Incorrecta**
Tipo	Generador de Secuencias
Condición de Prueba	Se debe cargar un programa que introduzca una instrucción que no pertenezca al Conjunto de Instrucciones, es decir, que tenga un OpCode no definido. (Ej. 1001).
Resultado Esperado	Se genera una Señal de Error de Instrucción no Válida.

Caso 2.2	
Nombre	**Administración de Pila Incorrecta**
Tipo	Generador de Secuencias
Condición de Prueba	Se debe cargar un programa que introduzca más llamadas a subrutinas o bucles anidados que el máximo de Niveles de Llamada permitidos por el ASIP, cuyo valor es TamañoPila + 1 (Por defecto, 9).
Resultado Esperado	Se genera una Señal de Error de Pila Rebosada.

Caso 3.1	
Nombre	**Administración de Cola Incorrecta**
Tipo	Controlador de Eventos
Condición de Prueba	Se debe cargar un programa que contenga muchas Instrucciones de Control seguidas. (Ej. Una instrucción MOV seguida de un bucle infinito).
Resultado Esperado	Se genera una Señal de Error de Cola Vacía cuando se intenta extraer una instrucción no existente de la Cola.

Se han realizado las Pruebas de acuerdo con las especificaciones anteriores y se ha verificado el comportamiento adecuado del ASIP en cada uno de los casos.

7. Implementación en un Soporte Físico

7.1 Descripción de la Aplicación Específica a Implementar

Como Aplicación a implementar se ha escogido el control de un monitor mediante una conexión VGA. Las salidas del ASIP se conectarán a la conexión VGA del soporte físico (las conexiones específicas se describen posteriormente en el apartado 7.5), y se ha cargado un programa de tal manera que se transmita video a la pantalla del monitor.

Para conseguir esto se deben enviar las señales de color de cada pixel así como las señales de sincronismo de la conexión VGA sincronizadas a unos tiempos determinados. Los datos específicos del protocolo VGA se describen posteriormente en el apartado 7.3.

La implementación del ASIP en físico requiere la conexión del ASIP a un microprocesador, el cual transmitirá al ASIP los datos de programa. Debido a esto se ha elegido como soporte físico un circuito integrado SoC Zynq™ (descrito en más detalle posteriormente en el apartado 7.5), que consiste en un microprocesador y una FPGA integrados en el mismo chip. El diseño hardware se ha actualizado para incluir la conexión al microprocesador mediante un bus AMBA.

Tras programar la FPGA con la especificación hardware, se programa el microprocesador con un simple código que escriba los datos del Programa del ASIP (previamente compilado con la Aplicación Ensambladora) al ASIP.

Una vez programados tanto el microprocesador como la FPGA, el sistema está listo para ser usado por el usuario. Tras pulsar los botones de Reset y Start, en ese orden, empieza a transmitirse el video a la pantalla.

Para la síntesis e implementación del hardware se ha utilizado el software Xilinx Vivado® Design Suite. Para la programación del microprocesador se ha utilizado el software Xilinx Software Development Kit.

7.2 Diseño Hardware

7.2.1 Interfaz AXI

Para poder conectar nuestro Diseño Hardware original del ASIP al microprocesador del dispositivo debemos tener en cuenta que el microprocesador se conecta con sus periféricos a través de un bus AMBA. Para conectar al bus AMBA, los periféricos deben tener una interfaz especial llamada AXI. Por tanto, debemos encapsular nuestro diseño original con esta interfaz AXI.

La versión de la interfaz AXI con la que cuenta el diseño es la versión AXI-4 Lite (La versión más reducida de la interfaz) para un dispositivo esclavo (como el ASIP, pues el maestro es el microprocesador). La interfaz AXI tiene por defecto 64Kdirecciones de memoria asignadas, de las cuales se usan sólo las cuatro primeras en la versión AXI-4, correspondiente a cuatro registros de 32 bits, identificados como slv_reg 0 a 3. Por lo tanto, el microprocesador sólo puede leer y escribir las cuatro primeras. De estos cuatro registros se usan los dos primeros, el primero para escribir la dirección de memoria del ASIP y el segundo para escribir los datos.

Estas 64K direcciones reservadas para el periférico AXI son parte del mapa de memoria completo del microprocesador, y se les asigna direcciones de memoria dentro del mapa de memoria de la placa. En la Tabla 18 se puede observar el mapeo de memoria del periférico, con valores por defecto si el ASIP es el único periférico del sistema.

Dirección	Nombre	Descripción
0x43C00000	Slv_Reg_0	Registro donde se guarda la dirección del dato
0x43C00001	Slv_Reg_1	Registro donde se guarda el valor del dato
0x43C00002	Slv_Reg_2	Registros no usados
0x43C00003	Slv_Reg_3	
0x43C00004 . . **0x43C0FFFF**	Resto de Memoria	Direcciones no Validas

Tabla 18: Mapeo de Memoria del Periférico AXI

Debido al número de entradas de la placa no se pueden usar todas las señales de Entrada Externa. Para esta aplicación se ajusta el valor del Parámetro Genérico NEntradas a 4 para considerar solo cuatro señales de Entrada Externa, pues este es el número de interruptores con los que cuenta la placa como se describirá en el apartado 7.5.

De la misma manera, la placa no cuenta con suficientes LEDs como para usar todas las señales de Error. Las Señales de Error son fijas y no dependen de los Parámetros Genéricos, por lo que las que no se consideren simplemente se ignoran y solo se hacen externas dos de ellas, las correspondientes a los errores más comunes: Error de Pila Rebosada y Error de Pila Vacía.

El bloque de diseño resultante se observa en la Figura 10.

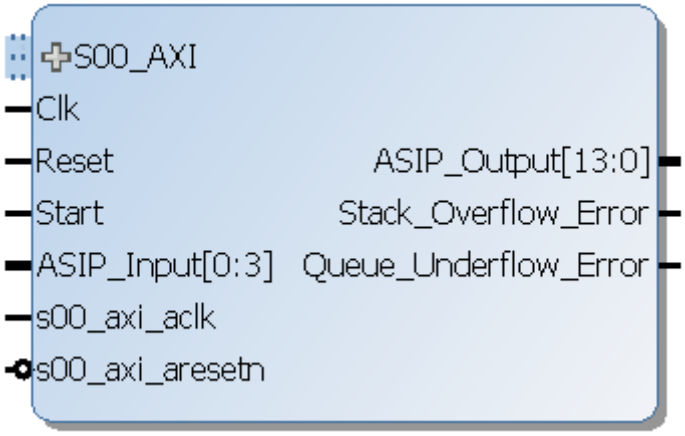

Figura 10: Diagrama Periférico AXI.

Se pueden observar las entradas del ASIP (que serán conectadas a botones e interruptores), así como sus salidas (conectadas al conector VGA y a los LEDs), acompañadas por tres otras señales, estas son las señales de comunicación con el microprocesador. Su función está descrita en la Tabla 19.

Señal	Nombre	Descripción
S00_AXI	Interfaz AXI	Bus de transmisión de datos
S00_axi_aclk	Reloj AXI	Señal de Reloj del bus AMBA
S00_axi_aresetn	Reset AXI	Señal de Reinicio Asíncrona del bus AMBA

Tabla 19: Señales de la Interfaz AXI.

El bus AMBA opera a una frecuencia de reloj de 100MHz fija, mientras que el ASIP opera con un reloj distinto, el reloj de la FPGA, que es configurable, por lo que se puede cambiar la frecuencia del ASIP si la aplicación lo requiere. Para esta aplicación, su frecuencia también es de 100MHz. Se debe considerar que la frecuencia mínima del ASIP debe ser 50MHz. Esto es debido a que para cada escritura en el ASIP se requieren dos datos escritos a través del bus AMBA (uno de dirección y otro de datos), y por lo tanto dos ciclos de reloj del bus por cada uno del ASIP. Valores menores provocarían que el bus AMBA escribiese datos más rápido de lo que se leen.

7.2.2 Conexión con el Microprocesador

Una vez establecida la interfaz AXI para el ASIP, éste se ha guardado como un IP (Intellectual Property), es decir, como un diseño hardware independiente.

Para conectarlo con el microprocesador se crea un diseño de bloques, una funcionalidad del software Vivado que permite conectar distintos IPs de manera rápida.

Primero se debe añadir un bloque ZYNQ, que representa el microprocesador desde el punto de vista de la FPGA. Las conexiones de este bloque determinan las conexiones de los periféricos de la placa física. Tras añadir el bloque ZYNQ al diseño se configura éste con la configuración por defecto para la placa sobre la que se implementará el diseño, la cual es una placa Digilent ZYBO que se describe en detalle posteriormente en el apartado 7.5. El bloque resultante se observa en la Figura 11.

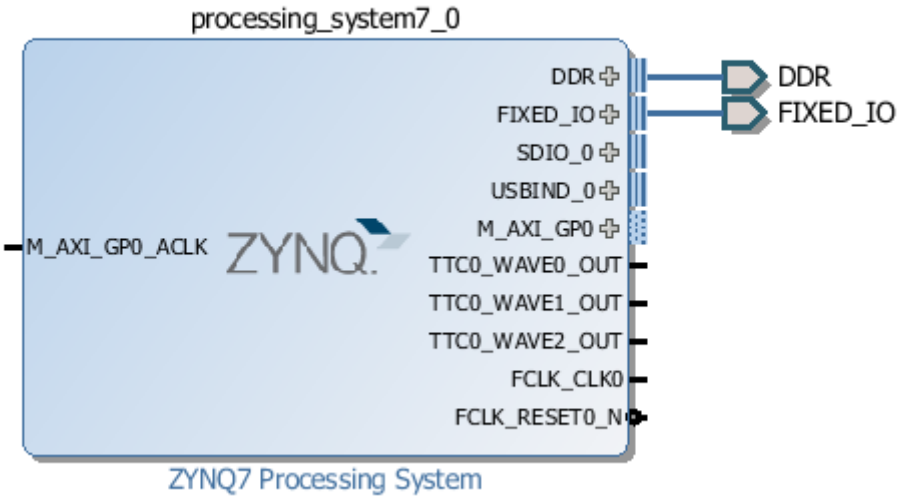

Figura 11: Diagrama del Procesador ZYNQ.

Se pueden apreciar en el bloque las distintas señales correspondientes a los periféricos de la placa como las interfaces USB y SD. Tras configurar el bloque ZYNQ se procede a añadir al diseño el periférico AXI que contiene nuestro ASIP. Una vez añadido, al tener una interfaz estándar, la herramienta Vivado automatiza la conexión. El diseño de bloques final se observa en la Figura 12.

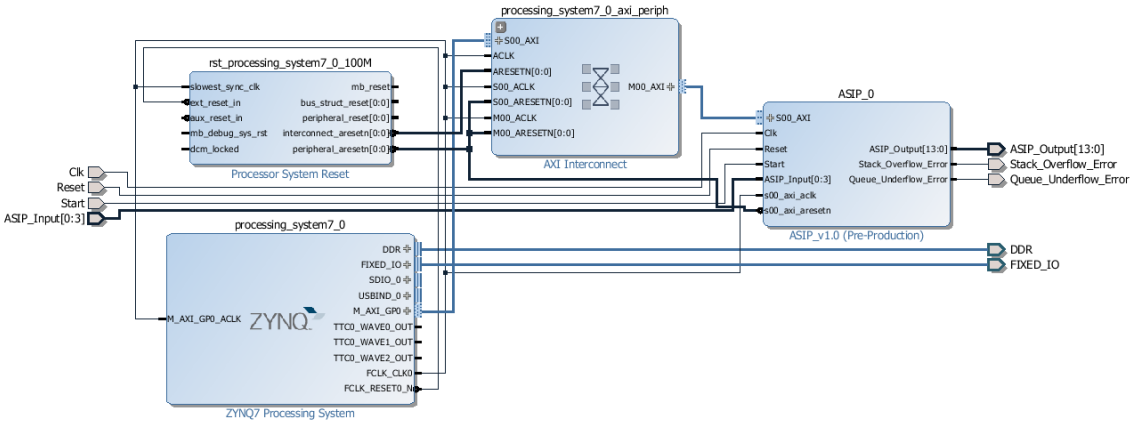

Figura 12: Diagrama de Bloques del Diseño Completo.

Podemos observar dos bloques nuevos, AXI Interconnect y Processor System Reboot. Estos dos bloques se añaden automáticamente al conectar un periférico AXI, siendo el primero la interfaz entre los periféricos AXI y el microprocesador, y el segundo el sistema que controla el reinicio del microprocesador.

Por último, se ha creado un Fichero de Restricciones “Constraints.xdc” que asigna los puertos de nuestro diseño a sus correspondientes pines de la placa. Las conexiones concretas se describen posteriormente en el apartado 7.3.

Tras realizar la síntesis, implementación, y generación del flujo de bits de programación de la FPGA, se exporta el hardware para su posterior programación.

7.3 Protocolo VGA

Antes de programar el diseño hay que tener en cuenta el protocolo que usa el conector VGA para transmitir datos, pues el programa se debe de escribir de acuerdo con éste.

La conexión VGA tiene un simple protocolo de barrido mediante señales de sincronización. Cuenta con dos señales de sincronización, sincronización horizontal (HSync) y sincronización vertical (VSync).Estas señales son activas a nivel bajo.

La activación de la señal HSync indicará la finalización de una fila de pixeles de la pantalla, y la señal VSync indicará la finalización de una pantalla completa. El tiempo entre activaciones y la duración del pulso de estas señales está determinado por la resolución deseada. Para esta aplicación se ha considerado una resolución de 640x480 pixeles con un ratio de refresco de pantalla de 60fps o imágenes por segundo.

Como se observa en la figura, cada línea horizontal tendrá una duración de T_s ciclos de reloj. Esta línea contará con un tiempo T_{disp} durante el cual se retransmitirán los pixeles de ésa línea, por lo que su duración es igual al número de pixeles por línea (en este caso 640), escribiendo un pixel por ciclo. Inmediatamente después se emite una secuencia de sincronismo horizontal compuesta por tres etapas, $T_{fp}, T_{pw}, y\ T_{bp}$ durante las cuales el color escrito será el negro (todas las señales de color inactivas). Durante T_{pw} la señal HSync estará activa (a nivel bajo). Se puede observar un simple esquema de esta temporización en la Figura 13.

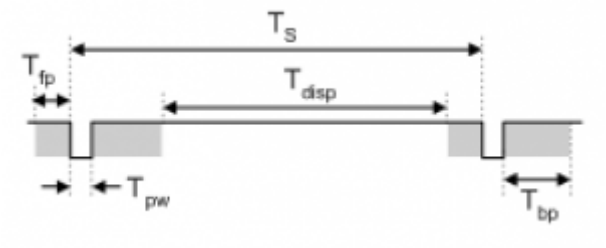

Figura 13: Sincronización de la conexión VGA.

La señal de sincronismo vertical VSync sigue el mismo funcionamiento, pero en vez de contar ciclos de reloj cuenta filas.

Para la resolución deseada la frecuencia del reloj debe ser de 25MHz, por lo tanto las duraciones de cada etapa son las descritas en la Tabla 20.

Etapa	Sincronismo Horizontal		Sincronismo Vertical		
	Ciclos	Tiempo(µs)	Líneas	Ciclos	Tiempo(ms)
T_s	800	32.00	521	416,800	16.672
T_{disp}	640	25.60	480	384,000	15.360
T_{fp}	16	0.64	10	8,000	0.320
T_{pw}	96	3.84	2	1,600	0.064
T_{bp}	48	1.92	29	23,200	0.928

Tabla 20: Tiempos de Sincronización de la conexión VGA.

7.4 Programación del Microprocesador

Teniendo en cuenta las especificaciones del protocolo VGA se procede a diseñar un programa sencillo, el Programa de Prueba adjunto en el Anexo 1. Este programa dibuja un patrón de ajedrez (cuadrados blancos y negros alternados) en la pantalla.

Debido a que el reloj de la placa se ha configurado a 100MHz por defecto, se elige un Valor de Prescalado de 4 para obtener una frecuencia eficaz de 25MHz.

Se rellena la lista de patrones que consiste de los colores deseados (en este caso blanco y negro) con ambas señales de sincronismo a nivel alto. A estos colores les sigue las señales de sincronismo, con todos los bits de color con valor 0 y la señal de sincronismo correspondiente también a 0. En total hay tres, una para el sincronismo horizontal, otra para el vertical, y una tercera para el tiempo que coinciden ambas.

Se rellena la lista de retrasos con los valores correspondientes a los patrones de color (en este caso, 32 ciclos debido a que los cuadrados son de 32x32 pixeles). A estos les siguen las duraciones de las etapas de sincronismo horizontal descritas en el apartado anterior.

Por último se programan las instrucciones, escribiendo instrucciones MOV con el color del pixel correspondiente (en este caso, alternando entre blanco y negro cada 32 ciclos) durante la duración de la línea horizontal, tras la cual se mandan instrucciones MOV con los patrones de sincronismo con sus correspondientes retardos. Se repite este grupo de instrucciones tantas veces como líneas haya en la pantalla, tras lo cual se repite lo mismo para las restantes tres etapas del sincronismo vertical. Durante estas tres etapas el color mandado es siempre negro.

El programa final “Main.c” usado se ha generado con el Ensamblador en el modo Periférico AXI.

Una vez se ha escrito el programa, se compila con la Aplicación Ensambladora, obteniendo una lista de pares dirección-dato en binario. Se abre el software Xilinx SDK y se importa el diseño hardware, tras lo que se crea en el SDK una nueva aplicación software que cuenta nada más con el reinicio del microprocesador. Al código de esta aplicación se le añade una función que escriba estos pares en sus correspondientes direcciones (0x43C00000 para la dirección y 0x43C00001 para el dato).

Una vez escrito el programa se programa la FPGA con el diseño hardware y el microprocesador con la aplicación software. Ambas operaciones se realizan a través de Xilinx SDK. Finalmente, el ASIP está listo para ser probado por el usuario.

7.5 Soporte Físico

La Implementación en Físico se realiza sobre una placa de desarrollo Digilent ZYBO Zynq™-7000 ARM/FPGA SoC [4].

La placa ZYBO cuenta con un SoC Zynq™-7000 de Xilinx, el cual se compone de un microprocesador ARM Cortex-A9 de dos núcleos, y una FPGA. La placa cuenta además con una serie de periféricos entre los que se incluyen dispositivos de entrada/salida simples (botones, interruptores, LEDs, etc.), puertos de comunicación (UART, USB, Ethernet, etc.) y puertos multimedia (VGA, HDMI, etc.).

Las conexiones que se utilizan en la implementación son las descritas en la Tabla 21.

Tipo	Nombre		Pin de la placa	Señal	Descripción
Interna	Reloj		L16	Clk	Señal de Reloj de la FPGA, configurada a 100Mhz.
Entrada	Botón 1		Y16	Reset	Botón de Reinicio Asíncrono
	Botón 2		V16	Start	Botón de Inicio de Ejecución
	Interruptores		T16	ASIP_Input[0]	Entradas Externas del ASIP
			W13	ASIP_Input[1]	
			P15	ASIP_Input[2]	
			G15	ASIP_Input[3]	
Salida	LEDs		M15	Stack_Overflow_Error	Indicadores de Error del ASIP
			M14	Queue_Underflow_Error	
	VGA	HSync	P19	ASIP_Output[13]	Puerto VGA al que se escribe la Salida de Datos del ASIP.
		VSync	R19	ASIP_Output[12]	
		R	F19	ASIP_Output[11]	
			G20	ASIP_Output[10]	
			J20	ASIP_Output[9]	
			L20	ASIP_Output[8]	
		G	F20	ASIP_Output[7]	
			H20	ASIP_Output[6]	
			J19	ASIP_Output[5]	
			L19	ASIP_Output[4]	
		B	G19	ASIP_Output[3]	
			J18	ASIP_Output[2]	
			K19	ASIP_Output[1]	
			M20	ASIP_Output[0]	

Tabla 21: Mapeado de los Puertos del Diseño a los pines de la Plataforma Física.

7.6 Pruebas Físicas

Las pruebas consistirán en la carga de un Programa de Prueba en el ASIP a través del microprocesador, siendo éste compilado mediante el Ensamblador en modo Periférico AXI. El programa está adjunto en el Anexo 1.

Requerimientos Previos:

Se debe conectar la Placa ZYBO al ordenador mediante un cable MicroUSB, y al monitor en el cual se desea visualizar el resultado mediante un cable VGA, como se observa en la Figura 14.

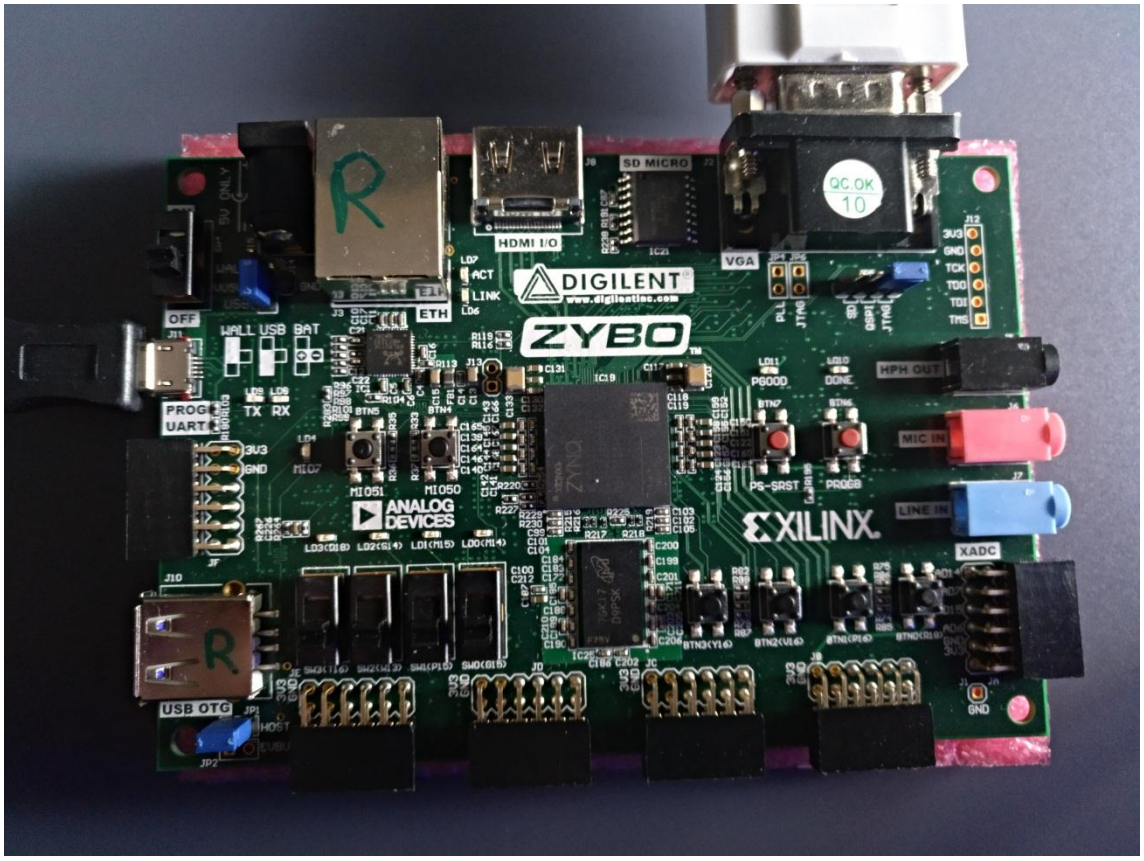

Figura 14: Conexión de la Placa

- **Programa de Prueba:**

  Una vez conectada la placa ZYBO como se describe en la Figura 14 se enciende y se programa mediante el Software Xilinx SDK. Primero se debe programar la FPGA y después programar el microprocesador con el programa del fichero Main.c generado por el Ensamblador a partir del Programa de Prueba. Una vez realizado este paso los indicadores de programación de la placa deben estar encendidos, como se observa en la Figura 15.

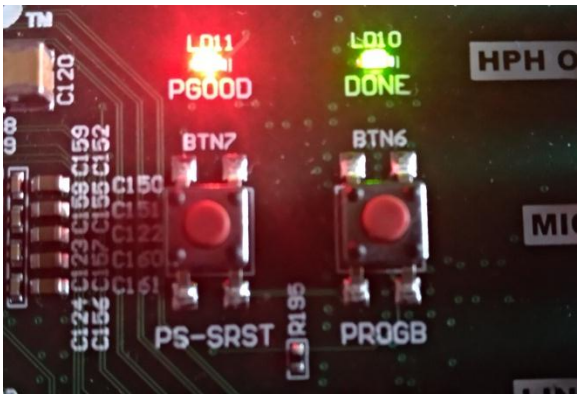

Figura 15: Indicadores de Programación de la Placa

Una vez cargado el programa se debe reiniciar el ASIP a su configuración inicial pulsando el Botón de Reinicio. Esto no afecta al programa cargado en el ASIP.
Tras reiniciar el ASIP se pulsa el Botón de START para iniciar la ejecución. La pantalla del monitor no debe mostrar nada.

Debido a la Instrucción SYNC del principio del programa, la ejecución no comenzará hasta que no se active el primer interruptor. Una vez activado éste, la pantalla del monitor presentará la imagen deseada, la cual se puede observar en la Figura 16.

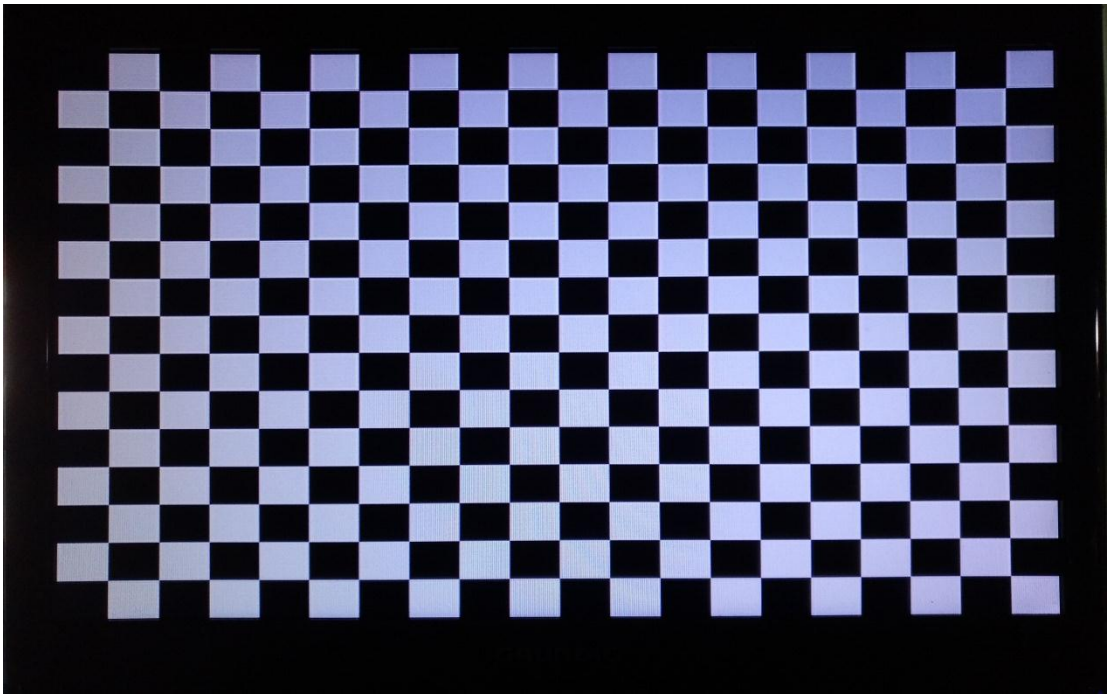

Figura 16: Resultado del Programa de Prueba

Los resultados de prueba han sido satisfactorios, se han obtenido los resultados esperados.

7.7 Resultados de Síntesis. Utilización de Recursos

Tras realizar la Síntesis Lógica del diseño podemos observar los recursos utilizados por el diseño, así como su consumo, su frecuencia de operación máxima, etc.

Los siguientes resultados de síntesis se han obtenido a partir del periférico AXI descrito en el apartado 7.2, para la plataforma física descrita en el apartado 7.5.

Recursos Utilizados:

Se pueden observar en la Tabla 22.

Recurso	Utilizado	Total	Utilizado (%)
LUT	313	17600	1.78
LUTRAM	24	6000	0.40
FF	291	35200	0.83
BRAM	0.5	60	0.83
BUFG	1	32	3.12

Tabla 22: Recursos Utilizados

Estadísticas de Consumo y Potencia:

Se pueden observar en la Tabla 23.

Consumo y Potencia	
Potencia Estática	0.105W
Potencia Dinámica	0.003W
Potencia Total	0.108W
Temperatura de Juntura	26.2ºC
Margen de Temperatura	58.8ºC

Tabla 23: Estadísticas de Consumo y Potencia.

Estadísticas de Temporización:

Se pueden observar en la Tabla 24.

Temporización	
Peor Margen Negativo (WNS):	3.158ns
Peor Margen Mantenido (WHS):	0.158ns
Peor Margen de Ancho de Pulso (WPWS):	3.75ns
Frecuencia Máxima Permitida:	146.15MHz

Tabla 24: Estadísticas de Temporización.

De estos resultados podemos observar que nuestro diseño consume muy pocos recursos lógicos comparado con la capacidad de una FPGA comercial moderna, y su consumo es bastante reducido.

El dato más importante de los Resultados de Síntesis es la Frecuencia Máxima Permitida, pues limita las aplicaciones que el ASIP puede realizar. El ASIP puede alcanzar una frecuencia de 146MHz, la cual es suficientemente alta para la mayoría de aplicaciones de generación de patrones.

8. Marco Regulador e Impacto Socioeconómico

8.1 Marco Regulador

Debido al carácter digital del proyecto, no se aplica ningún tipo de normativa regulatoria o legislación sobre el desarrollo del proyecto. Sin embargo, se deben de tener en cuenta los estándares técnicos de las tecnologías usadas.

En este proyecto se ha usado el Lenguaje de Descripción de Hardware VHDL, el cual es un estándar IEEE, siendo la última versión el estándar IEEE 1076-2008. De la misma manera, se han tenido en cuenta estándares para la programación en VHDL, como el estándar IEEE 1164 que consiste en el tipo de datos “std_logic” usado para la gran mayoría de señales de este diseño VHDL. De esta manera se ha normalizado el proyecto de acuerdo con los estándares internacionales.

En adición al uso de estos estándares, las herramientas usadas para el desarrollo e implementación del ASIP, Xilinx Vivado Design Suite y Xilinx Software Development Kit, son herramientas comerciales de uso extendido, y que permiten exportar el diseño como IP en formatos de uso extendido, lo que facilita la compatibilidad del diseño con otros sistemas. De la misma manera, VHDL es un Lenguaje de Descripción de Hardware de uso extendido en Europa.

8.2 Impacto Socioeconómico

La principal opción de comercialización de este diseño es la comercialización como IP del ASIP. Éste IP debe ser personalizable para elegir entre el IP por sí mismo, o su versión como periférico AXI. De esta manera el diseño puede ser utilizado por varias otras personas o compañías, ya sea por sí mismo o como parte de un diseño superior.

La comercialización como IP implicaría ingresos cada vez que éste sea utilizado por otro, con ningún coste de materiales o mantenimiento asociado. El coste total del ASIP por tanto es igual al coste inicial del desarrollo, sin ningún gasto a largo plazo.

Para poder elegir un precio de venta del diseño debemos tener en cuenta el coste inicial del desarrollo. Se puede aproximar el coste medio de una hora de trabajo de un ingeniero en España a 35€, ya incluidos costes de seguridad social y costes indirectos. Para obtener el coste total del desarrollo se deben estimar las horas trabajadas en el proyecto y multiplicarlas por el coste de la hora de ingeniero para obtener el Coste Total del diseño. Los resultados obtenidos se observan en la Tabla 25.

Mes de Trabajo	Horas Semanales	Horas Mensuales	Coste Mensual
1	**12**	**48**	**1.680 €**
2	**12**	**48**	**1.680 €**
3	**12**	**48**	**1.680 €**
4	**20**	**80**	**2.800 €**
5	**30**	**120**	**4.200 €**
		Coste Total	**12.040 €**

Tabla 25: Coste del Desarrollo

El coste medio de la licencia de un IP de alta demanda está en torno a los 5000€, mientras que el coste de un IP como el de este diseño sería más cercano a los 2000€ por licencia. Con esta estimación podemos obtener el número de licencias mínimo que debemos vender para recuperar la inversión inicial.

$$Nº\ de\ Licencias = \frac{Coste\ Final}{2000€} = \frac{Coste\ Final}{2000€} = 6.02\ \approx 6\ licencias$$

Luego para recuperar la inversión inicial debemos vender un mínimo de 6 licencias para cubrir costes. A partir de 6 licencias vendidas el IP genera beneficios.

9. Conclusiones

9.1 Trabajo Realizado

Se ha completado la realización de todos los objetivos del proyecto, habiendo aumentado el alcance de alguno de ellos. Los objetivos completados son los siguientes:

- **Se ha diseñado el ASIP en VHDL.**
  Se han cumplido todos los requerimientos. El diseño es altamente personalizable debido a los parámetros genéricos y se puede exportar como IP para uso en otros diseños

  Funciones adicionales añadidas:
  - Se ha añadido la funcionalidad de las Señales de Error.

- **Se ha desarrollado una Aplicación Ensambladora para la programación del ASIP.**
  La aplicación compila correctamente los programas a Código de Máquina.

  Funciones adicionales añadidas:
  - Se ha añadido la opción de generar directamente el código fuente C para su uso en la programación del ASIP a través de una placa SoC.
  - Se ha añadido una serie de comprobaciones en el compilador que indican la presencia de errores en el programa a compilar.

- **Se ha validado el diseño mediante Simulación.**
  Se han considerado una serie de Casos de Prueba y se han realizado pruebas del comportamiento del ASIP para todos estos casos. El resultado de todas las pruebas ha sido satisfactorio y el ASIP funciona correctamente.

  Funciones adicionales añadidas:
  - Debido a la adición de las Señales de Error al diseño del ASIP, se ha realizado una segunda batería de pruebas que comprueba que las señales se generan correctamente cuando se detecta un error. El resultado de todas las pruebas ha sido satisfactorio.

- **Se ha implementado el diseño en un Soporte Físico y se han realizado Pruebas Físicas sobre éste.**
  Se ha implementado el diseño del ASIP como periférico AXI en una placa SoC. Se ha establecido la conexión entre el ASIP y el microprocesador de la placa de tal manera que se pueda programar el ASIP directamente mediante software. Se ha cargado un programa de prueba y se ha comprobado su correcta escritura en la memoria interna del ASIP y su correcta ejecución. Se ha realizado un breve estudio de los recursos consumidos por el ASIP, así como los límites de temporización del diseño.

Funciones adicionales añadidas:

- Debido al carácter de alta frecuencia del ASIP, la comprobación de la correcta generación de las señales de salida una a una se debe realizar mediante un analizador lógico. Para facilitar la comprobación del ASIP se han conectado las señales de salida al conector VGA de la placa, conectando éste a un monitor, pudiendo comprobar su correcto funcionamiento mediante la observación de la imagen presentada por el monitor. De esta manera también se observa el ASIP en una aplicación real.

9.2 Posibles Mejoras

Debido al cumplimiento de todos los Objetivos de Proyecto y la inclusión de funcionalidades adicionales hay pocas áreas de mejora del diseño:

- **Optimización del Diseño:** Se puede optimizar el diseño para reducir aún más su utilización de recursos, así como aumentar su frecuencia de operación máxima. Una leve optimización podría incrementar la frecuencia máxima (146.15MHz) a 150MHz, permitiendo al ASIP ser usado en un rango de aplicaciones más amplio.

- **Exportación como IP Configurable:** Se puede exportar el diseño como IP y establecer una interfaz de configuración reducida que calcule los parámetros genéricos automáticamente en base a las necesidades del usuario. De esta manera se reduce el número de parámetros que el usuario debe configurar y se facilita el uso del ASIP.

Bibliografía

1. K. Keutzer, S. Malik y A.R. Newton. *From ASIC to ASIP: The Next Design Discontinuity.* IEEE International Conference on Computer Design: VLSI in Computers and Processors. (2002)
2. M. Kumar Jain, M. Balakrishnan y A. Kumar. *ASIP Design Methodologies: Survey and Issues*. Department of Computer Science and Engineering. Indian Institute of Technology, Delhi. (2000)
3. IEEE. *IEEE Standard VHDL Language Reference Manual.* (2009).
4. Digilent, Inc. *ZYBO™ FPGA Board Reference Manual.* (2017).

Anexo 1: Programa de Prueba

Programa de Prueba

```
#Prescale
4

#Pattern Table (14 señales):
; EventID HSync VSync RRRR GGGG BBBB
0       00000000000000      ; Reset
1       11000000000000      ; Negro
2       11111111111111      ; Blanco
3       01000000000000      ; BH-CH
4       10000000000000      ; BV-CV

#Delay Table
0       100
1       32
2       14
3       98
4       48
5       640

#Code
MOV 0, 0
SYNC 0, 1
Inicio:

CALL Imagen  ; Pinta la imagen en pantalla
NOP                 ; NOP en delay slot
JUMP Inicio  ; Repite infinitamente

Imagen:
       LOOP 7                    ; Cubre 7*(32+32) 448 filas
              CALL Fila_par
              LDC 32             ; Recorre 32 filas
              CALL Fila_impar
              LDC 32             ; Recorre 32 filas
       DEC
       CALL Fila_par
       LDC 32              ; Recorre 32 filas (32 + 448 = 480 filas total)
       CALL VSync          ; Patron de sincronismo vertical (imagen finalizada)
       NOP
DECRET Imagen                          ; Retorno de subrutina

Fila_par:
       LOOP 10             ; Recorre 1 fila = 10 x (32 + 32) = 640 pixel
              MOV 1, 1     ; 32 pixel en negro
              MOV 2, 1     ; 32 pixel en blanco
       DEC
       CALL HSync          ; Patron de sincronismo horizontal (fila finalizada)
       NOP
DECRET Fila_par

Fila_impar:
       LOOP 10             ; Recorre 1 fila = 10 x (32 + 32) = 640 pixel
              MOV 2, 1     ; 32 pixel en blanco
              MOV 1, 1     ; 32 pixel en negro
       DEC
       CALL HSync          ; Patron de sincronismo horizontal (fila finalizada)
       NOP
DECRET Fila_impar

HSync:
       MOV 1, 2
       MOV 3, 3
```

```
       MOV 1, 4
DECRET HSync

VSync:
       LOOP 10
              MOV 1, 5
              CALL HSync
              NOP
       DEC
       LOOP 2
              MOV 4, 5
              MOV 4, 2
              MOV 0, 3
              MOV 4, 4
       DEC
       LOOP 29
              MOV 1, 5
              CALL HSync
              NOP
       DEC
DECRET VSync

#END
```

Anexo 2: Ficheros de la Aplicación Ensambladora

Ejemplo de Código de Máquina.

```
10000101000000000000000000000000000000000000100
10000000000000000000000000000000000000000000000
10000000000001000000000000000000011000000000000
10000000000010000000000000000000111111111111111
10000000000011000000000000000000001000000000000
10000000000100000000000000000000010000000000000
10000100000000000000000000000000000000000000001
10000100000001000000000000000000000000000100000
10000100000010000000000000000000000000000001110
10000100000011000000000000000000000000001100010
10000100000100000000000000000000000000000110000
10000100000101000000000000000000000000001010000000
00000000000000000000000000000000000000000000000000
00000000000001000000000000000000011000010000000
00000000000010000000000000000001100000000000101
00000000000011000000000000000001110000000000000
00000000000100000000000000000001110000000000010
00000000000101000000000000000001010000000000111
00000000000110000000000000000001100000000010000
00000000000111000000000000000001000000000100000
00000000001000000000000000000011000000000010111
00000000001001000000000000000001000000000100000
00000000001010000000000000000010000000000000000
00000000001011000000000000000011000000000010000
00000000001100000000000000000001000000000100000
00000000001101000000000000000011000000000100010
00000000001110000000000000000001110000000000000
00000000001111000000000000000001010000000000101
00000000010000000000000000000001010000000001010
00000000010001000000000000000000000000001000001
00000000010010000000000000000000000000010000001
00000000010011000000000000000010000000000000000
00000000010100000000000000000011000000000011110
00000000010101000000000000000001110000000000000
00000000010110000000000000000001010000000010000
00000000010111000000000000000001010000000001010
00000000011000000000000000000000000000010000001
00000000011001000000000000000000000000001000001
00000000011010000000000000000010000000000000000
00000000011011000000000000000011000000000011110
00000000011100000000000000000001110000000000000
00000000011101000000000000000001010000000010111
00000000011110000000000000000001110000000000000
00000000011111000000000000000000000000000011000011
00000000100000000000000000000000000000001000100
00000000100001000000000000000001010000000011110
00000000100010000000000000000001010000000001010
00000000100011000000000000000000000000001000101
00000000100100000000000000000011000000000011110
00000000100101000000000000000001110000000000000
00000000100110000000000000000010000000000000000
00000000100111000000000000000001010000000000010
00000000101000000000000000000000000000100000101
00000000101001000000000000000000000000100000010
00000000101010000000000000000000000000000000011
00000000101011000000000000000000000000100000100
00000000101100000000000000000010000000000000000
00000000101101000000000000000001010000000011101
00000000101110000000000000000000000000001000101
00000000101111000000000000000011000000000011110
00000000110000000000000000000001110000000000000
00000000110001000000000000000010000000000000000
00000000110010000000000000000001010000000100010
```

Ejemplo de Programa C para el Microprocesador.

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xbasic_types.h"

Xuint32 *baseaddr = (Xuint32*)(XPAR_ASIP_0_S00_AXI_BASEADDR);

void write_program();
int to_binary(char string[32]);
void write_line(int addr, int data);
void read_line(char buff[255]);

int main(){
      init_platform();
      write_program();
      print("Programación del ASIP completada.\n");
      cleanup_platform();
      return 0;
}

void write_program() {
      *(baseaddr)      = 0b00000000000000000010000101000000;
      *(baseaddr + 1)  = 0b00000000000000000000000000000100;
      *(baseaddr)      = 0b00000000000000000010000000000000;
      *(baseaddr + 1)  = 0b00000000000000000000000000000000;
      *(baseaddr)      = 0b00000000000000000010000000000001;
      *(baseaddr + 1)  = 0b00000000000000000011000000000000;
      *(baseaddr)      = 0b00000000000000000010000000000010;
      *(baseaddr + 1)  = 0b00000000000000000011111111111111;
      *(baseaddr)      = 0b00000000000000000010000000000011;
      *(baseaddr + 1)  = 0b00000000000000000001000000000000;
      *(baseaddr)      = 0b00000000000000000010000000000100;
      *(baseaddr + 1)  = 0b00000000000000000010000000000000;
      *(baseaddr)      = 0b00000000000000000010000100000000;
      *(baseaddr + 1)  = 0b00000000000000000000000000000001;
      *(baseaddr)      = 0b00000000000000000010000100000001;
      *(baseaddr + 1)  = 0b00000000000000000000000000100000;
      *(baseaddr)      = 0b00000000000000000010000100000010;
      *(baseaddr + 1)  = 0b00000000000000000000000000001110;
      *(baseaddr)      = 0b00000000000000000010000100000011;
      *(baseaddr + 1)  = 0b00000000000000000000000001100010;
      *(baseaddr)      = 0b00000000000000000010000100000100;
      *(baseaddr + 1)  = 0b00000000000000000000000000110000;
      *(baseaddr)      = 0b00000000000000000010000100000101;
      *(baseaddr + 1)  = 0b00000000000000000000001010000000;
      *(baseaddr)      = 0b00000000000000000000000000000000;
      *(baseaddr + 1)  = 0b00000000000000000000000000000000;
      *(baseaddr)      = 0b00000000000000000000000000000001;
      *(baseaddr + 1)  = 0b00000000000000000110000010000000;
      *(baseaddr)      = 0b00000000000000000000000000000010;
      *(baseaddr + 1)  = 0b00000000000000001100000000000101;

}
```